



Norwegian University of  
Science and Technology

# Ship Control by Autonomous Tugboats

Development of an Adaptable Test Platform  
for Autonomous Tugboat Research and  
Development

**Sondre Næss Midtskogen**

Master of Science in Mechanical Engineering

Submission date: June 2017

Supervisor: Martin Steinert, MTP

Co-supervisor: Kristoffer Slåttsveen, MTP

Norwegian University of Science and Technology  
Department of Mechanical and Industrial Engineering





---

# Acknowledgements

I would like to take this opportunity to show my gratitude to all the people who have made this project possible.

Firstly, I would like to thank my partner in crime, Rebecca Lillian Cox, without whom neither my tugboats or I would have found our way through this project.

Secondly, I would like to thank my supervisor, Professor Martin Steinert, who have created an amazing space for innovation in TrollLABS, seething with exciting projects and people, and who provided me with this amazing project and the means for handling it. I want to thank my co-supervisor, Kristoffer Sltsveen, for the personal guidance he has offeret throughout the year, and also all the other people who comprise the unique environment that is TrollLABS.

I want to thank Kongsberg Maritime, for supporting and showing genuine interest in this project, providing me with additional motivation to succeed.

I also want to thank my good friend, Kenneth, who helped me proof-read this thesis during the final hours.

I want to thank the employees at the department of Material MTP for allowing me to set up a huge pool and occupy a large space in the workshop for more than a month.

Finally, I want to thank my girlfriend, Ingvild, whose mutual love and support is a daily source of energy and motivation, particularly during the hectic final weeks of this thesis.

I could probably have done this without you.

But I wouldn't want to.

---

---

---

# Abstract

This project originated as an open-ended product development challenge on the topic of autonomous tugboats, supported by Kongsberg Maritime. The vision behind the project is twofold. In part, it is a futuristic vision of swarms of autonomous tugboats inhabiting the world's harbours. The other part is to show that by applying the PD tools and methods taught at TrollLABS, we can make significant headway in an emerging field of development, using very little time and resources.

The focus of this project has been to develop and realize a realistic, scale test setup for implementation and validation of different strategies for ship control by autonomous tugboats, and to demonstrate a working proof of concept strategy on this setup in collaboration with Rebecca Lillian Cox. The test setup is low-cost, easy to use and can adapt to experiment with an array of different parameters pertaining to autonomous tugboats.

Because this is a master's thesis in product development, the methods and tools used to manage the project have been an equally important part of the thesis. The most important tools and methods used are adaptations of Concurrent Engineering, Design For X and Agile, and the main motivation behind the choice of these has been to keep a quick pace throughout the project, without making design mistakes that result in major rework.

While the scope of the project forced the author to take many shortcuts in technical implementation, often at the expense of performance and technical depth, the learning outcome in terms of product development management has been tremendous. Having acted as mechanical, electrical, control and software engineer, manufacturer and product manager in a single project, he has gained a unique perspective on the work-flow and interaction between these fields. He considers this experience invaluable in a world where the fields of engineering are more interwoven than ever.

This, the functional test setup and a study on autonomous tugboat operations have been the main results of this project.

---

---

# Sammendrag

Dette prosjektet begynte som en pen produktutviklingsutfordring p emnet ”autonome taubter”, stttet av Kongsberg Maritime. Prosjektets visjon er todelt: Den ene delen er en fremtid-srettet visjon av svermer av autonome taubter i verdens havner. Den andre delen er vise at ved bruke produktutviklingsverkyt og -metoder som undervises ved TrollLABS, kan vi gjre signifikant fremskritt i et nyoppsprunget fagfelt, selv med veldig lite tid og ressurser.

Hovedoppgaven i dette prosjektet har vrt utvile og realisere en skalert testplattform for implementering og validering av forskjellige strategier for styre skip ved hjelp av autonome taubter, og demonstrere en ”proof of concept” strategi p denne platformen i samarbeid med Rebecca Lillian Cox. Testplattformen er lavbudsjett, enkel bruke og kan brukes til eksperimenterer med et antall forskjellige parametre som er relevant for autonome taubter.

Da dette er en masteroppgave i produktutvikling, har metodene og verkytene som har blitt brukt for hndtere prosjektet, vrt en like viktig del av oppgaven. De viktigste verkytene som er brukt er adaptasjoner av *Concurrent Engineering*, *Design For X* og *Agile*, og hov-edmotivasjonen bak valget av disse har vrt holde rask fremgang gjennom hele prosjektet, uten beg designfeil som resulterer i betydelig ekstraarbeid.

Til tross for at omfanget av prosjektet har tvunget forfatteren til ta mange snarveier i den tekniske implementasjonen, ofte p bekostning av ytelse og teknisk dybde, har lring-sutbyttet innen produktutviklingsmetode vrt utrolig. Det ha hatt rolle som mekanisk, elektrisk, regulerings- og programvareingenir, prosjektleder og produksjonsarbeider i ett og samme prosjekt, har gitt ham et unikt perspektiv p arbeidsflyten og interaksjonen mel-lom disse feltene. Han anser denne erfaringen som uvurderlig i en verden der fagfelt stadig knyttes nrmere sammen.

Dette, en fungerende testplattform og et studie av autonome taubtoperasjoner har vrt de viktigste resultatene i dette prosjektet.

---

# Table of Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Sammendrag</b>	<b>v</b>
<b>Table of Contents</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>xii</b>
<b>Abbreviations</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Summary of pre-master's thesis . . . . .	1
1.2 Problem description . . . . .	2
1.3 The control problem . . . . .	3
1.4 Outline . . . . .	6
<b>2 Work method</b>	<b>9</b>
2.1 Concurrent Engineering . . . . .	10
2.2 Design For X . . . . .	11
2.3 Agile . . . . .	11
<b>3 Test setup development</b>	<b>13</b>
3.1 Requirements . . . . .	13
3.2 Electronic hardware . . . . .	15
3.2.1 Components . . . . .	15
3.2.2 Circuit . . . . .	17
3.3 Physical design . . . . .	20
3.3.1 Material and manufacturing method . . . . .	21

---

3.3.2	Design . . . . .	21
3.4	Manufacturing and assembly . . . . .	25
3.5	Software . . . . .	26
3.5.1	Controller design . . . . .	26
3.5.2	ROS . . . . .	29
3.5.3	Final program flow design . . . . .	29
3.6	Combining the subsystems . . . . .	35
3.6.1	Problems and bugs . . . . .	36
3.6.2	Visualization . . . . .	38
<b>4</b>	<b>Test setup FAT</b>	<b>41</b>
4.1	Waypoint tracking . . . . .	41
4.2	Contact control . . . . .	43
4.3	Initiating contact with ship . . . . .	43
4.4	Surrounding the ship . . . . .	43
4.5	Controlling the ship . . . . .	45
<b>5</b>	<b>Project evaluation and further work</b>	<b>47</b>
5.1	Work methods . . . . .	47
5.2	Test setup . . . . .	48
5.2.1	Evaluation . . . . .	48
5.2.2	Further work . . . . .	48
5.3	Ship control . . . . .	49
<b>6</b>	<b>Conclusion</b>	<b>51</b>
	<b>Bibliography</b>	<b>53</b>
	<b>Appendices</b>	<b>55</b>
<b>A</b>	<b>Test setup source code</b>	<b>57</b>
<b>B</b>	<b>Risk assessment</b>	<b>107</b>
<b>C</b>	<b>Pre-master thesis</b>	<b>113</b>



# List of Tables

3.1	Product demand specification . . . . .	14
3.2	Electronic component list . . . . .	16
3.3	Score-based decision-making on material and manufacturing method. . .	22

---

# List of Figures

1.1	A tugboat exerting force on a ship by pushing against its hull . . . . .	3
1.2	Control system of a tugboat pushing against a surface . . . . .	4
1.3	A tugboat exerting force on a ship by pushing against its hull . . . . .	5
1.4	Line-of-Sight Guidance (green) vs. Trajectory Tracking (blue) . . . . .	6
3.1	Two-level prototype circuit board, from pre-master’s thesis. . . . .	17
3.2	”Tugcar” prototype, from pre-master’s thesis. . . . .	17
3.3	Prototype circuit diagram, using D-SUB connectors, which were not realized. The four wires at the bottom of the figure go to the load cell. . . . .	18
3.4	Battery connector versions 1 and 2 . . . . .	19
3.5	Circuit board generation 2 . . . . .	19
3.6	Printed circuit board. The design is the same as in Generation 1, with the addition of the two LEDs and the general-purpose pad grid. . . . .	20
3.7	Minion, physical prototype generation 1 . . . . .	23
3.8	Physical prototype generation 2. . . . .	25
3.9	Initial software design . . . . .	28
3.10	Final program flow design . . . . .	30
3.11	Computer Vision is used to find the pose of the boats . . . . .	32
3.12	The Simulation node neatly replaces all the nodes which interface with the real world, which means the rest of the program can run normally. The figure is only used to illustrate this point; details of the diagram are the same as in Fig. 3.10 . . . . .	34
3.13	Sequence diagram showing tugboat and waypoint handling between modules. Note that the Waypoint Manager in this sequence diagram is my placeholder module, which I made in order to test my own program, before implementing Cox path planning. The inner workings of this module will therefore differ from that used in the final product, but the interactions with other modules remain unchanged. . . . .	35
3.14	Macro shot of the whole test setup. . . . .	37
3.15	Retro-fitted keel mounted to pre-existing holes, to improve motion stability. . . . .	38

---

3.16	The two visualization channels produced by the setup. Each boat has a unique colour. . . . .	39
4.1	Two tugboats tracking a path from one corner of the setup to the other, two different runs. . . . .	42
4.2	By using a smartphone to scan this QR code, the reader will taken to a video of a similar waypoint-tracking run on the test setup. . . . .	42
4.3	Tugboats surrounding ship in order to gain control. Time line progresses from upper left to lower right. . . . .	44
4.4	By using a smartphone to scan this QR code, the reader will taken to a video of tugboats surrounding the ship and achieve control. . . . .	45
4.5	By using a smartphone to scan the QR code on the left, the reader will taken to a video of 5 tugboats controlling the position of the ship to a contained area arund the ship’s waypoint for 5 consecutive minutes. The waypoint is indicated by the red arrow going from the ship to a static point in the middle of the left half of the videofeed. The image to the right is the path taken by all boats during the video. The ship’s path is marked by a red curve. The teal tugboat was broken down during the session, which can be seen in the video. . . . .	46

---

# Abbreviations

SCAT	=	Ship Control by Autonomous Tugboats
PD	=	Product Development
LSG	=	Line-of-Sight Guidance
TT	=	Trajectory Tracking
FAT	=	Factory Acceptance Test
CE	=	Concurrent Engineering
DFX	=	Design For X

---

# Introduction

The challenge for this project was conceived at TrollLABS, an innovation research center run by Professor Martin Steinert, as an open-ended product development challenge on the topic Autonomous Tugboats. The project is sponsored by Kongsberg Maritime, a world leading company on autonomous marine vessels.

The vision behind the project is twofold. In part, it is a futuristic vision of swarms of autonomous tugboats inhabiting the worlds harbours . The other part is to show that by applying the PD tools and methods taught at TrollLABS, we can make significant headway in an emerging field of development, using very little time and resources.

Defining the scope, focus, and ultimate goal of the project was part of the challenge, and has been the main focus of the authors pre-master's thesis (Fossen, 2002).

## 1.1 Summary of pre-master's thesis

The fleet of tugboats, particularly harbour tugs, is increasing both in number, due to a rising demand for their services, and in size, due to the high cost and scarcity of good crew. Because harbour tugs primarily perform the routine task of guiding and assisting large vessels within a pre-determined space, they are ripe for autonomization, and therefore the focus of this project.

To reduce the workload of the project, some limitations had to be defined. The most important are:

1. No higher-level operations such as harbour logistics, route planning or obstacle avoidance, are considered. The goal was defined as being able to precisely manoeuvre a large ship using autonomous tugboats.
2. The tugboats only apply force on the ship by pushing against its hull no tug lines are involved.

The problem was approached with a prototyping mindset. A simple simulation environment for rapid testing of control algorithms was set up, and served as a digital equivalent

to cardboard prototyping. This allowed the author to quickly grasp the essence of the problem, and get a feel for what would work and what would not.

Due to the strict requirements for predictability and precision, and the well-defined problem, advanced control theory, like that used in dynamic positioning systems, was preferable to swarm technology or machine learning for this problem. However, these fields of study could still be used in higher or lower-level problems, such route planning (higher) or plant identification (lower). An in-depth study of advanced control theory ensued, both by general control theory subject learning, and by reviewing more directly relevant literature.

Towards the end of the project, work started on planning a physical setup on which different control algorithms could be implemented and tested. Preliminary electronic components to perform the various lower-level functions were selected and tested in critical function prototypes.

The main takeaways from the pre-master's project are subject knowledge and a concrete plan for the master's thesis, formulated as a set of research problems.

## 1.2 Problem description

I was fortunate enough to get a partner for the master's project, Rebecca Lillian Cox. In her master's thesis, she will address route planning, both for the ship and for each individual tugboat. This allows me to focus on developing the test setup and implementing ship control, and reduces uncertainty around tugboat movement and unwanted interference from tugboats. Cox path-planning software will be treated as a black box, with an interface we will design in close collaboration.

In light of this, and after getting some perspective on the project thesis, the research problems from the pre-master's project were re-evaluated. With the assistance of Cox path-planning software, I will:

1. *Realize a realistic scale test setup for implementation and validation of different strategies for ship control by autonomous tugboats (SCAT).*
2. *Demonstrate a working proof of concept strategy on this setup.*

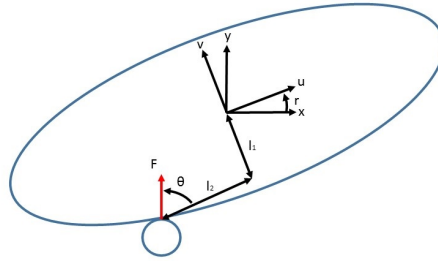
The official assignment text contains a third research problem:

3. *Develop, implement, test and evaluate different strategies for SCAT, in the physical test setup and/or a simulation environment, and make recommendations for real-size implementation of SCAT.*

During the semester, it became evident that research problems 1 and 2 were more than ambitious enough for a master's project. Research problem 3 was subsequently removed from the problem definition for this, but remains as the intended use-case for the test setup described in research problem 1.

This project differs from existing research by approaching the problem from the practical side, rather than developing a specific, advanced theoretical approach, and try to verify it in a carefully controlled, tailor-made environment or simulation. The aim of this





**Figure 1.1:** A tugboat exerting force on a ship by pushing against its hull

project is to facilitate rapid research and development of SCAT, by producing a set-based prototype (See appendix C) in which to test out different approaches and solutions to the problem, and study interesting parameters and interactions, with the instant learning feedback and unique insights only a realistic setting can produce. In addition, it is the aim to contribute to this R&D through the insights made during development and early use of this setup, and to suggest the way forward based on this.

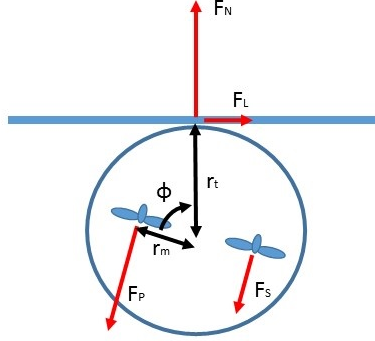
In the pre-master's thesis, I argued for a control theory based approach to the problem, over swarm intelligence and machine learning based approaches. This was not because the latter two are inherently *bad* choices for this problem – on the contrary, I strongly believe they will have their place in the field of autonomous tugboats. However, the control theory approach is the more predictable and comprehensible of the three, which makes it the best foundation for developing the setup, and for early experimental implementation. For the scope of this thesis, the problem will be treated purely as a traditional guidance, navigation and control problem, and the setup will be tailored to this approach.

### 1.3 The control problem

The motion of a marine surface vessel, when moving at low speed, is normally described by equations 1.1 (Fossen, 2002)

$$\dot{\eta} = \mathbf{R}(\psi)\boldsymbol{\nu} \quad (1.1a)$$

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{D}\boldsymbol{\nu} = \boldsymbol{\tau}, \quad (1.1b)$$



**Figure 1.2:** Control system of a tugboat pushing against a surface

where

$$\tau = \mathbf{B}\mathbf{u} \quad (1.2a)$$

$$\eta = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \quad (1.2b)$$

$$\nu = \begin{bmatrix} u \\ v \\ r \end{bmatrix} \quad (1.2c)$$

Here,  $\mathbf{R}$  is the rotation matrix,  $\mathbf{M}$  is the inertia matrix,  $\mathbf{D}$  is the damping matrix,  $\mathbf{B}$  is the configuration matrix for the thrusters and  $\mathbf{u}$  is the control input for each thruster.

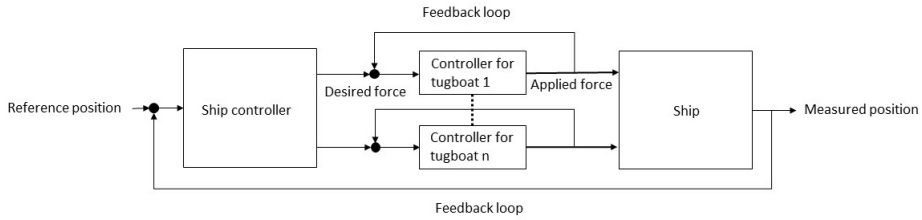
The movement of the ship will be controlled by a number of tugboats, which exerts force on a point on the ship by pushing against its hull, as depicted in **Fig. 1.1**. By modelling the force applied by the tugboat as a thruster, we get the thruster configuration matrix

$$\mathbf{B} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \cos(\theta)l_2 - \sin(\theta)l_1 \end{bmatrix} \quad (1.3)$$

Here we assume that the change in the tugboats impact points is negligible from one control iteration to the next, and model the tugboats as periodically stationary thrusters. The angle  $\theta$  can be assumed static, or be used as a control variable. This is how tugboats are modelled in the literature reviewed in the pre-master's thesis.

While this model works well for designing ship control algorithms, it fails to account for the steps required to deliver these control outputs, which is no trivial task. In fact, this is a moderately complex control problem in its own right.

Let us take a closer look at the tugboat depicted in **Fig. 1.1**, and assume it is designed as suggested in the pre-master's thesis, with two fixed motors in parallel, on opposite sides



**Figure 1.3:** A tugboat exerting force on a ship by pushing against its hull

of the tugboats centre of mass, as depicted in **Fig. 1.2**. In order to provide the correct force vector,  $\mathbf{F}$ , the tugboat must balance the thrust from each motor to provide the normal and lateral forces that comprise  $\mathbf{F}$ . These are given by

$$F_N = (F_P + F_S)\cos(\phi) \quad (1.4a)$$

$$F_L = \frac{(F_P - F_S)r_m}{r_t} \quad (1.4b)$$

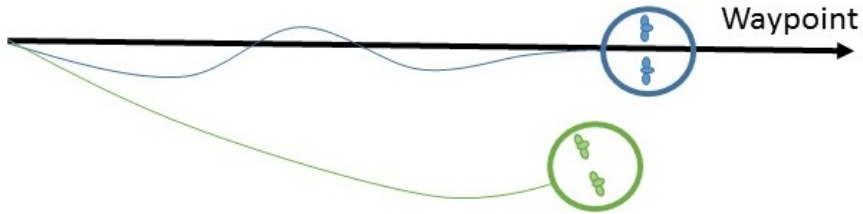
Where  $F_P$  and  $F_A$  are the thrust forces exerted by the port and starboard thrusters, respectively. For an electric tugboat, this is in turn given by functions of the voltage applied to each motor.

In the pre-master's, a two-tier controller was proposed to address this problem, where the higher-level controller (ship controller) outputs setpoints for the lower-level controller(s) (tugboat controller(s)), like in **Fig. 1.3**. In this configuration, the higher-level controller must be robust enough to handle deviations from the requested controller output.

Another nontrivial task is to get to the point where the tugboats are arranged in a desired manner around the ship, ready to assume control. Two elements need to be in place for this to happen. The first is simply to define this desired arrangement of tugboats. Ideally, this arrangement should result in a force closure (the ability to exert force in any direction), but how, or even if this is achieved, will differ from algorithm to algorithm, and will therefore be included as an element of the ship control algorithm.

The second is how each tugboat should get to its desired point. Unless an unobstructed straight line exists between the tugboat and the end goal, this requires some sort of path planning. In Cox' thesis, she will handle this problem by breaking down the path into a set of waypoints, with an unobstructed straight line between each consecutive point. Following the given path from waypoint to waypoint falls to me, and requires a different controller.

There are two main approaches to this problem: Line-of-Sight Guidance (LSG) and Trajectory Tracking (TT) (Fossen, 2002). LSG is by far the simpler of the two to implement, but is also the most susceptible to external influence. This is because of how the heading controller is implemented. In LSG, the heading controller is simply designed to make the boat face towards the next waypoint at all times. In TT, the boats heading is



**Figure 1.4:** Line-of-Sight Guidance (green) vs. Trajectory Tracking (blue)

adjusted if the boat deviates from the desired path. When subjected to external forces, TT will attempt to work against these, to keep the vessel on a desired path, while LSG will not. This is illustrated in **Fig. 1.4**. In both methods, speed can be controlled by a simple open-loop controller, or by any number of more advanced feedback controllers.

Thus, the problem can be broken into five steps, each of which is required to successfully demonstrate a proof of concept ship control algorithm on the test setup. These five steps comprise the test setups factory acceptance test (FAT), the results of which will be an indication of my performance on this project. The steps are:

1. *Tugboats must be able to go to a given point.*
2. *Once physical contact with the ship is established, contact must be maintained.*
3. *Tugboats must be able to initiate contact with the ship, that is, changing from 1 to 2 dynamically.*
4. *Several tugboats must be able to perform 3 in unison, to collectively achieve force closure, or otherwise gain control over the ship.*
5. *Once 4 is achieved, the tugboats must be able to control the motion of the ship, through a combined effort.*

Additionally, the test-setup must be able to handle miscellaneous unintentional behaviour, such as a tugboat failing to make contact with the ship, or a temporary loss of control for any reason.

## 1.4 Outline

It is important to state that this is first and foremost a master's thesis in product development, and this will greatly effect the form of the thesis. The focus is roughly equally divided between *what* has been done, and *how* it has been done. This includes how the product development project has been managed, and which work methodologies and product development tools have been applied to the different problems, and to what effect. Although the author has some background in control theory beyond what is expected of

a mechanical engineering student, I am not a control engineer, and this is not a thesis in control theory.

In chapter 2, I will describe the most important PD tools and methods I have used to manage this project, and briefly describe how I intend to use them. This is the main theoretical foundation for the thesis. In chapter 3, I will document the process of developing the test setup, grounded in the foundation described in chapter 2, from the rough plan at the conclusion of my pre-master's thesis, to a functional proof of concept prototype. In chapter 4, the test setup will be subjected to a factory acceptance test based on the five steps discussed above, and its performance will be briefly discussed. In chapter 5, the work methods and the test setup will be evaluated, and I will discuss my thoughts on further work on the project, based on what I have learned over the course of this year. Finally, in chapter 6, I will wrap up the different aspects of the project, and evaluate the project as a whole.



# Chapter 2

## Work method

The product development (PD) part of this project lies in developing and manufacturing a physical test setup, capable of performing different ship control algorithms like those discussed in the pre-master's thesis (See appendix C). This spans the entirety of a PD project, culminating in a software environment that must pass a FAT, and small-scale mass production of tugboat drones.

Ulrich and Eppinger (Ulrich, 2003) divides a PD project into six main stages:

1. Planning
2. Concept development
3. System-level design
4. Detail design
5. Testing and refinement
6. Production ramp-up

By this model, the pre-master's project spanned stage 1 and 2, and ended somewhere between stage 3 and 4. Detail design have begun on some areas of the project like selecting and procuring preliminary electronic components while most areas are still in the system-level design stage.

The pre-master's project was largely a discovery project, with the purpose of mapping the problem- and solution space. At this point, the problem space is mostly defined, and a rough image of the solution space is formed. This means that I must switch from hunter mode to gatherer mode (Steinert and Leifer, 2012), and require a different set of tools.

This is a large project for one person to complete in just a few months. To finish in time, corners must be cut, assumptions must be made, napkin calculations replace rigorous mathematical analysis, and simplicity and time efficiency are major decision variables. However, if any of this cause significant design flaws, these flaws may propagate through

the design process before they are discovered, at which point they require a lot of rework to fix.

The methods and tools I will use are focused on minimizing lead time by making quick, but grounded design decisions, without resulting in significant rework. The most important are *Concurrent Engineering*, *Design For X*, and *Agile*.

## 2.1 Concurrent Engineering

Concurrent engineering is a nonlinear project management methodology which takes all life-cycle phases of a product into consideration at the early design stage (Prasad, 1996; Xue et al., 1999). Unlike the traditional linear models like that presented above, CE embraces the fact that requirements and dependencies that go upstream, as well as downstream, by including all parties involved with a PD project, across disciplines, in the early design phase. This has two main advantages:

1. The PD lead time is greatly reduced, because no department has to wait for another to finish their work before they can start with their own.
2. Rework is reduced because errors are passed on and discovered immediately, rather than at the end of a stage. For example, a part designed in a manner which makes it impractical to manufacture, will be spotted by manufacturing before the part is refined, or dependent subsystems are designed.

The concurrent engineering methodology assumes cross-functional teams, with different areas of expertise. Being alone on this project, some adaptation is required.

The way I approach concurrent engineering in this project, is by organizing my work in short design loops in which I touch all aspects of the project. In each loop, I locate the pain points, which can be tasks with long lead times, such as procuring components, or systems with many dependants. Their dependencies are identified across the project, and I focus on resolving these. Each loop, new pieces fall into place, the prototypes are slightly more refined, I learn a little more, and the project is finished step by step. I work on a fluid time plan, in which the current loop is planned in detail, while the subsequent loops are less detailed, and more adaptable.

A good example of how I have used Concurrent Engineering is with the propulsion system. The propulsion system generates several requirements and limitations across the board, such as motor control, controller design and lower hull design, and should therefore be settled early in the project. The propulsion system is dependent on the inertial properties of the tugboats, to give the tugboats the required manoeuvrability.

In order to make a decision on propulsion design, rough prototypes of the physical tugboat design, circuitry and motor control software had to be implemented. This meant that I started looking at manufacturing by the third week of the semester, rather than at the very end, at which point a redesign of the propulsion system would have required massive rework of several dependent systems.



## 2.2 Design For X

A big problem in concurrent engineering, is to get people with different backgrounds to share a common understanding of what is the best design. [...] *unless one can provide a basis for discussion that is grounded in quantified cost data and systematic design evaluation, directions will often be dictated by the most forceful individual in the group, rather than being guided by a knowledge of the downstream results* (Boothroyd, 1994).

Design for X (DFX) is one of the most effective approaches to implementing concurrent engineering in practise (Eastman, 2012). DFX is an umbrella term for a host of design tools like Design for Assembly and Design for Manufacturing. A DFX tool focuses on a few factors important for a single life-cycle aspect, X, and provides a systematic approach to evaluate design decisions with respect to this. Different DFX tools can be applied to different products or subsystems in a product, or to different stages in the development of a single product.

I will not use these tools slavishly in my project. With a production volume of 10 tugboats or less, the time investment required to systematically learn and follow several DFX tools is simply not feasible. Instead, I extract the core wisdom from each tool used, and use this in my design process, whenever applicable.

I use Design For Manufacturing and Design For Assembly in designing the physical aspects of the tugboats, to minimize production time, and Design for Modularity in software design, to make the system more adaptable, both during and after development.

## 2.3 Agile

A prototype is any representation of parts of a product used to demonstrate or explore some aspect of the product (Houde and Hill, 1997). This includes any temporary or completed versions of the products subsystems. From this viewpoint, the final product is nothing but the coming together of all these prototypes into a whole. It follows that prototypes can be used as the means for driving the project forward, by continuously developing, refining and combining prototypes.

In the world of software development, this methodology is embodied by *Agile* (Fowler and Highsmith, 2001). The Agile Manifesto is a collaboration between proponents of the many "light" iterative and incremental development methodologies that were becoming popular during the 1990s. These methods came as a response to the increasingly rapidly changing requirements and tough competition in the software market, as a response to the "heavy" linear methodologies like the Waterfall model (Larman and Basili, 2003). The manifesto for Agile software development is based on 12 principles, 3 of which are about early and continuously producing working software. The same principles were applied to mechanical product development in 1986 (Hirotaka and Ikujiro, 1986), described as the "rugby method". Here, working software is replaced with prototypes. The main advantages of using this methodology are

- Reduced lead time: Agile is all about getting things done. A product that works is produced very quickly, then iterated on until it works well.

- **Adaptability:** Because the product consists of many independently working modules, changes can be made by altering or replacing one or more of these modules, without affecting the remainder of the product.

Consequently, Agile might be seen as rushed, and may miss out on the *best* outcome due to insufficient mapping of the design space. However, I am not looking for *best* I am looking for *functional*. Because of the time constraints on this project, a slight improvement in performance is not worth a substantial increase in development time. Further, whatever the best outcome is at the start of the project, changing requirements and new insights will likely mean that this outcome is not best by the end of the project. By keeping the end goal adaptable, I can converge on a result that at the very least is *useful*.

# Test setup development

The test setup consists of subsystems in three major fields; Electronic hardware, physical design and software. While these subsystems are developed simultaneously, using methods from Concurrent Engineering, they are presented separately for the sake of readability, and to show how different tools are used for different problems.

The Agile methodology sets working software or hardware over exhaustive documentation, which is necessary to get the rate of progress required to finish this project. Nevertheless, all critical decisions and pivoting points are documented. This includes the major prototype iterations, which for physical design and electronic hardware means working prototypes used for iterative learning, and in developing software.

For software, iteration is much harder to show, as it is a continuous process with thousands of small steps. Instead, the initial plan and the reasoning behind it will be presented, before describing the final version in more detail.

## 3.1 Requirements

The most fundamental requirements for the test setup comes from the use case, which is to carry out ship control algorithms like those discussed in the pre-master's thesis. The setup should be robust and scalable, to allow users to test vastly different control algorithms. For example, algorithms using only a few tugboats should work the same as algorithms with large swarms of twenty or thirty tugboats.

In the pre-master's thesis, some basic product requirements were extracted from these use cases, in order to start planning the test setup. For the readers convenience, a refined and reworked version of these requirements are presented in table 3.1, in a more comprehensive form. The numbers in this table are not carved in stone. They are educated guesses based on what I learned during the pre-master's project, and what I know of the different technologies involved.

An equally important requirement is the deadline for the master's thesis. As have been mentioned several times already, this is an ambitious project for a master's thesis, which means there is no time to waste on over-engineering. It is therefore important to state

that I am not striving for perfect, I am simply working for good enough. Further, some requirements must be set for tugboat manufacturing. It is difficult at this point to predict how much time is required to produce a tugboat, but at the very least, active production time per tugboat should not exceed one day, and I should have 6 tugboats ready by the end of project week 13 (the week after Easter), to have enough time to test and improve the setup and develop a proof of concept Ship Control algorithm.

No.	category	Metric	Value
1.1	<b>Overall requirements</b>	Must pass FAT	
1.2		No. of tugboats	$\geq 6$
1.3		No. of tugboats must be easy to increase	
1.4		Preferable if a simulation is developed	
2.1	<b>Necessary information</b>	Absolute position and orientation of ship	$\pm 5$ cm, 5 deg
2.2		Position and orientation of tugboats relative to ship	$\pm 5$ cm, 5 deg
2.3		Pushing force applied by each tugboat, at bow	$\pm 0.01$ N
3.1	<b>Tugboat movement</b>	Degrees of freedom	3 (x, y, o)
3.2		Speed, acceleration roughly equivalent to that of a duck swimming leisurely in a pond	
4.1	<b>Tugboat, physical</b>	Projected shape	Cylindrical
3.2		Diameter	10-20 cm
3.2		Electronics protected from water	
3.2		Reasonably stable in water; should never capsize under normal operation	
4.1	<b>Tugboat, electrical</b>	Battery life	$\pm 10$ min
3.2		2-way wireless communication on each tugboat and main computer. Range:	$\pm 5$ m
4.1	<b>Ship control</b>	Must be interchangeable	
4.2		Preferable if MATLAB and Simulink can be used	

**Table 3.1:** Product demand specification

## 3.2 Electronic hardware

The electronic components are the starting points around which the rest of the test setup are built. The mechanical design is made to house these components, and the software is designed to interface with them. Therefore, and because of the long procurement time, these were selected during the pre-master's project, subjected to *critical function prototyping*, and procured at the very start of the master's project. The components that performed well for each function are presented below.

The main design criterion for the electronic hardware, beyond performing these functions, is to minimize the time spent on development, production and maintenance, both on the hardware itself and on driver software. I use *Design For Manufacture and Assembly* and *Design For Maintainability* when designing the electronic hardware.

### 3.2.1 Components

I quickly decided not to make a custom PCB from scratch, as this would have been too time consuming in the design phase. Instead, I put together a collection of breakout boards, which are quicker and easier to use. With a batch size this small, the increase in assembly time and cost is outweighed by the reduction in design time. The final list of components is presented in Table 3.2.

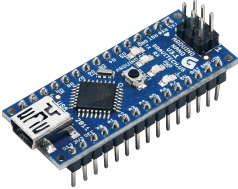



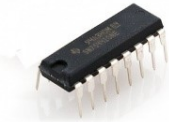

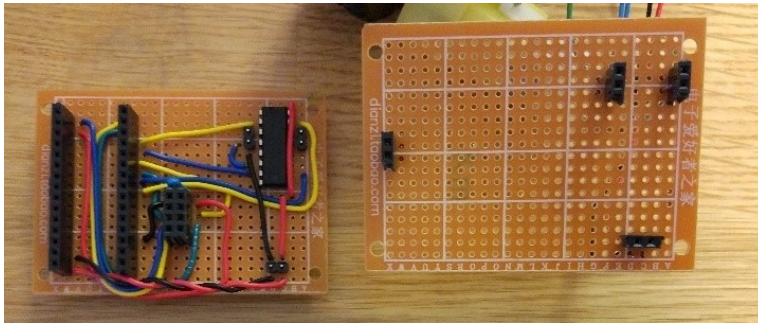
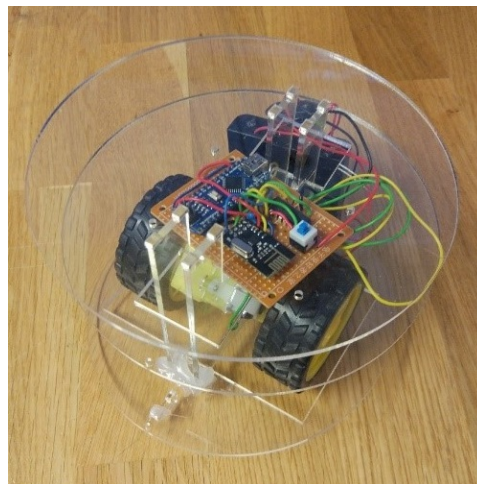
Component	Description	Comment
<b>Arduino Nano</b> 	Microcontroller development board. 16MHz, 32kB program memory, 2kB flash memory. 20 I/O pins, PWM, SPI, I2C ++	Very easy to program and interface with other components. Supports many useful libraries. Smaller than Arduino Uno, and can be soldered onto a PCB
<b>Nrf24l01+</b> 	2.4 GHz multi-channel radio transceiver, 250 kbps 2Mbps transfer rate, SPI, auto retransmit and acknowledgement	Good library for interfacing with Arduino exists. Can send and receive at the same time, making timing easier.
<b>Load cell + HX711</b> 	1 kg Wheatstone bridge load cell and 24-Bit ADC with Wheatstone bridge interface.	Good library for interfacing with Arduino exists. Very high resolution (30 g/Bit).
<b>Brushed DC motor</b> 	High-torque motors with 2mm shaft. Operating voltage 1.5-12V, 3000 5400 rpm ++	Small, cheap, but powerful DC motors with long shaft. Can operate under water, at a range of voltages.
<b>SN75441one</b> 	Quadruple Half-H Driver. Operating current up to 1A per driver (2A in bursts), voltages between 4.5 and 36V.	Simple to use, works well with the selected motors.
<b>Batteries</b> 	12V, 1100 mAh MiNa batteries taken from old power drills.	Scavenged from the mechatronics workshop in-house. On-hand, easy to charge saves time and money. Quite large and heavy, which gives the tugboats inertial stability, but demands greater hull size.

Table 3.2: Electronic component list



**Figure 3.1:** Two-level prototype circuit board, from pre-master's thesis.

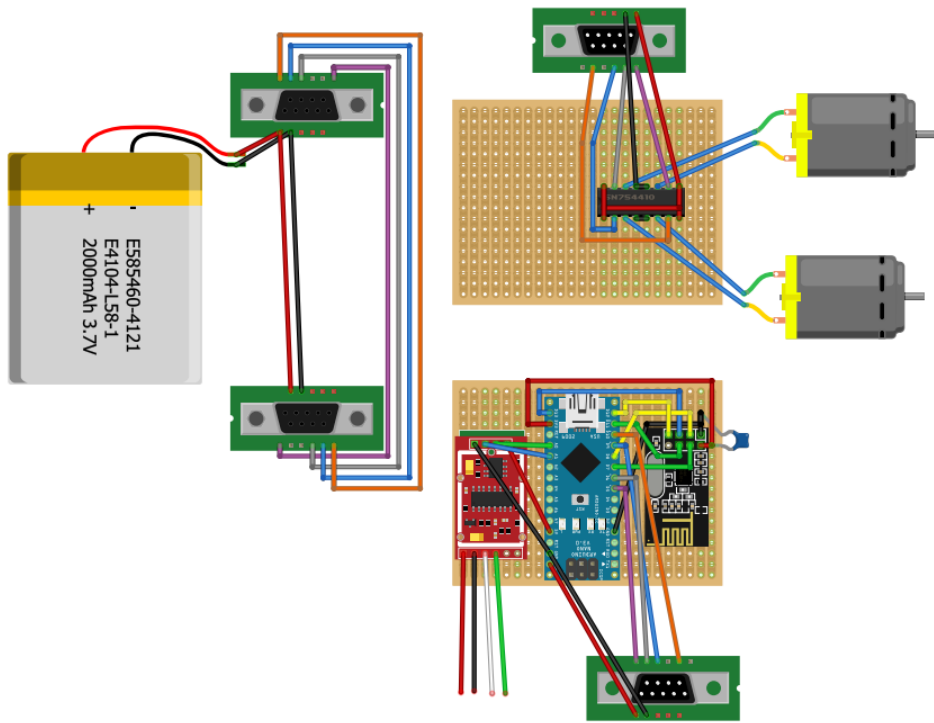


**Figure 3.2:** "Tugcar" prototype, from pre-master's thesis.

### 3.2.2 Circuit

From the pre-master's project, I had a two-level prototype circuit board, see Fig 3.1. The right-hand board, which contains all the breakout boards, is completely detachable from the tugboat, to facilitate alterations during development. It is mounted on the left-hand board via four sets of male-to-female header pins. The left-hand board is attached to the tugboat, and contains the battery pack and the two motors.

This design requires a large opening above the circuit board, which in a submerged tugboat implies that it must be mounted horizontally beneath the top, like in the tugcar prototype, see Fig. 3.2. With the arrival of the huge drill batteries into the design, this central space becomes occupied. But two unused, narrow spaces in the design are revealed on either side of the battery. This warrants a redesign of the circuit boards, to allow them to be slid into narrow cavities.



**Figure 3.3:** Prototype circuit diagram, using D-SUB connectors, which were not realized. The four wires at the bottom of the figure go to the load cell.

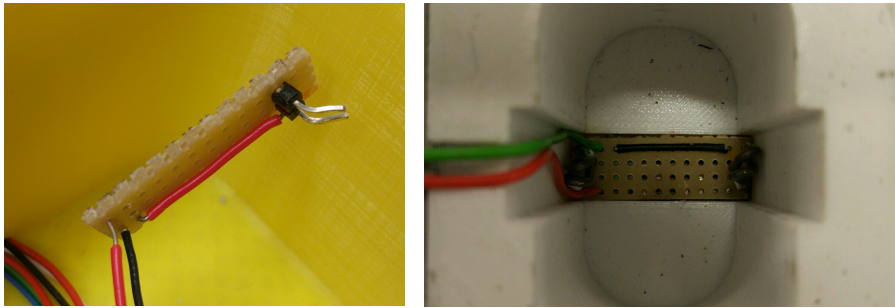
### Generation 1

This iteration of the circuit board was produced to operate Minion, the first tugboat prototype, and the load cell and amplifier was introduced to the circuit. As Minion's hull design featured two circuit board slots, the secondary board from Generation 1 was placed in its own slot, and the motor controller was moved to this board to make room for the load cell amplifier on the main board. With this design, the secondary circuit board has room for additional sensors if this turns out to be needed, and/or a different motor controller setup in case the motors need to be changed or upgraded.

This iteration also introduced the drill battery to the circuit. A quick battery connector was made using a small piece of prototyping board and two pairs of male header pins. This setup worked, but was far from durable, as can be seen in Fig. 3.4a.

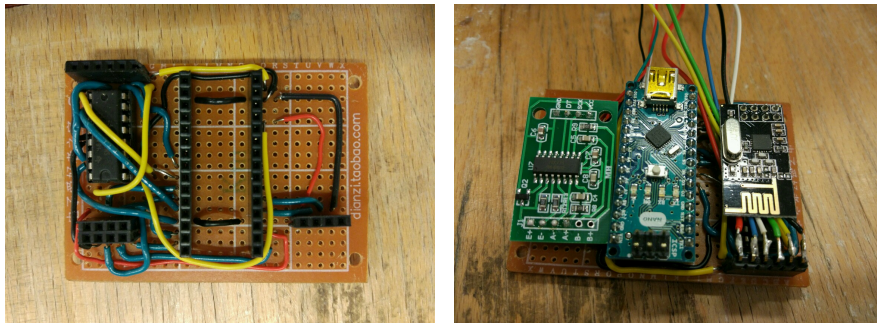
The plan was to connect the two boards and the tugboat using D-SUB connectors, see Fig. 3.3, but these turned out to be too time consuming to assemble. For the sake of progress, Minion was assembled with non-detachable circuit boards. Further, the cavities were too small to fit components mounted on female header pins, forcing me to solder components directly onto the circuit board. This turned out to be a bad idea, as the Arduino broke during testing, but not before I had tested the most important aspects.





(a) First battery connector prototype. One set of header pins missing. (b) Final battery connector, made with small steel coils

**Figure 3.4:** Battery connector versions 1 and 2



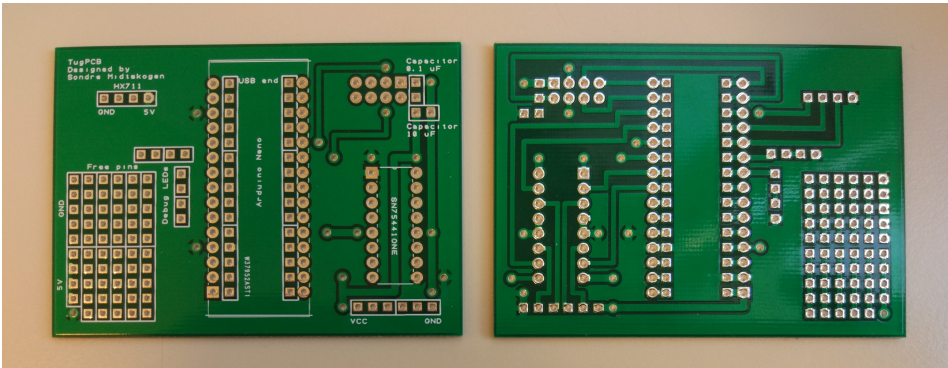
(a) All components are removable (b) All components on a single board.

**Figure 3.5:** Circuit board generation 2

## Generation 2

After getting more experience with the capabilities and limitations of the 3D printers used for manufacturing, I was able to open up the PCB slots enough to move back to components mounted on female header pins again. This opened up room for the motor controller below the radio transceiver, and I was able to fit all the components on a single circuit board, resulting in fewer external connections, see Fig. 3.5.

I soldered the wires from motors and battery to a row of male header pins, which connected to a female equivalent on the circuit board as can be seen in Figs. 3.5. This design was both easier to assemble and took less space than using D-SUB connectors. This way, the circuit board is once again completely detachable from the tugboat, which makes replacing damaged components possible. The battery connector was also upgraded to a more durable design, using a small coil of steel wire, see Fig. 3.4b.



**Figure 3.6:** Printed circuit board. The design is the same as in Generation 1, with the addition of the two LEDs and the general-purpose pad grid.

### Generation 3

Satisfied with the design of Generation 2, I decided to have PCBs professionally made. This greatly reduces the number of parts, which is the most important point in Design For Assembly (Boothroyd, 1994). It took approximately 12 hours to design, verify and order a batch of 10 PCBs. This PCB reduced the total assembly time for electronics from approximately six hours to a little over one hour, meaning I save time after only three tugboats.

On the PCB, I added a pair of LEDs for debugging purposes, which would turn out to be invaluable. I also added pads connected to each of the Arduinos pins, and a small section of prototyping pads, to make the design more adaptable to changing requirements and further use. The PCB is depicted in Fig. 3.6.

## 3.3 Physical design

The physical design of the tugboat is simply a hull to house the electronics, and keep most of them dry. To this end, I want to keep the bottom and the sides sealed, with the only opening at the top of the tugboat. The top also has to be flat, to accommodate an Aruco marker, a requirement set by the software modules.

Only the load cell and the motors have direct spatial constraints. The load cell must be placed at the very front of the tugboat, while the motors must be placed in a manner which allows for forward thrust as well as momentum. For best effect, they are placed in parallel, symmetrical about the longitudinal axis, with propellers aligned with the centre of mass.

The remaining design constraints are the conflicting needs for a sufficient and stable buoyancy and a diameter between 10 and 20 cm. The centre of mass must lie directly below the centre of buoyancy, to keep the tugboat upright in the water. This implies that the battery, which comprises the majority of the weight, must be placed near the centre of buoyancy, and preferably symmetrical along the longitudinal axis to maintain lateral symmetry. This leaves two voids in the design space on either side of the battery, suitable

for circuit boards. Thus, an optimal rough spatial configuration of the components has been established by following the design constraints.

### 3.3.1 Material and manufacturing method

With this in place, the next step is to determine a material and manufacturing method. This is the most important design decision on the physical design, as it has the greatest impact on the total lead time of the tugboats. Based on the machines and equipment available to me, a list of possibilities was made and evaluated. Each plausible combination of material and manufacturing are rated with a score from 1 to 5 on 5 different criteria, where 5 is best score. Active production time, being the most important criterion, is weighed doubly. The criteria are:

1. **Cost:** Total impact on department resources: Material, machine time and staff time costs.
2. **Design freedom:** Design time is assumed inversely proportional to the degree to which the design is constrained by the option.
3. **Iteration:** How much work is required to make changes between iterations.
4. **Active production time:** Estimation of how much hands-on time is required for manufacturing and assembly.
5. **Confidence:** How confident I am with using this option.

Ultimaker/Prusa type FDM (Fused deposition modelling) 3D printing scored well across the board, beating the Objet PolyJet type printer by material cost and machine availability. There are seven such 3D printers at the department, while there is only one Objet, and the material is much cheaper.

3D printing is a very slow manufacturing process, but very little of that is hands-on time, as it requires no moulds, lengthy setup procedure or surveillance. 3D printing can produce almost any shape imaginable, and iterating is just a matter of changing the CAD file and generate a new g-code file. Finally, I have some experience with 3D printing, and I can use the printers without any instructions or supervision.

### 3.3.2 Design

Having settled for FDM 3D printing as manufacturing method, I must *Design For Manufacturing*. This method scored a 4 on textitDesign freedom because there are some small, yet important, manufacturing constraints.

While the Objet PolyJet 3D printer can produce almost any shape by filling voids with a dissolvable support material, the simpler FDM 3D printers like Ultimaker and Prusa are a bit more limited. They too can produce support structure, but in this case made of the same material as the main part. This is less precise and reliable than with the PolyJet, and is often hard and time consuming to remove after production. Therefore, I want to avoid using support material if possible.

Manufacturing method	Material	Cost	Design freedom	Iteration	Active prod. Time (weight 2x)	Confidence	Sum
Laser cutting	Acrylic	4	2	5	3	4	21
Ultimaker/Prusa FDM 3D printing	PLA or ABS plastic	4	4	5	5	5	28
Objet PolyJet 3D printing	Photopolymer "VeroWhitePlus"	1	5	5	5	5	26
In-house injection moulding	Different plastics	5	3	3	1	2	15
Out-of-house injection moulding	Different plastics	4	3	1	4	1	17
Milling	POM or wood	1	3	4	3	3	17
Vacuum forming	Acrylic	5	1	3	2	2	15
Vacuum forming + Laser cutting	Acrylig and/or MDF	4	4	3	1	3	16

**Table 3.3:** Score-based decision-making on material and manufacturing method.

The reason support material is used in 3D printing is because the next layer of material needs to be built on top of the previous layer. If there is nothing to build upon, the material is ejected into thin air. This will not always be a problem, particularly for small areas with large tolerances. But by avoiding any flat horizontal overhangs, one can make sure that each new strip of material is built at least partially on top of the previous layer, eliminating these spaghetti surfaces.

Further, each print must have a firm footprint on the print bed, or risk print errors due to vibrations in the part, warping, or the whole coming loose. Finally, 3D printing is still a developing manufacturing method, and the Prusa or Ultimaker printers are not high-end machines, so the risk of a print failure is very real.

Considering all this, and the fact that I want detail work both on the top and bottom of my tugboats, I want to make them in several parts, each with a solid flat footprint. While this is directly contrary to DFAs tenet of minimizing the number of parts, it reduces both the active and passive manufacturing time, by eliminating the need for support material and reducing the number of wasted print hours due to prints failing after a long time.

### Generation 1, "Minion"

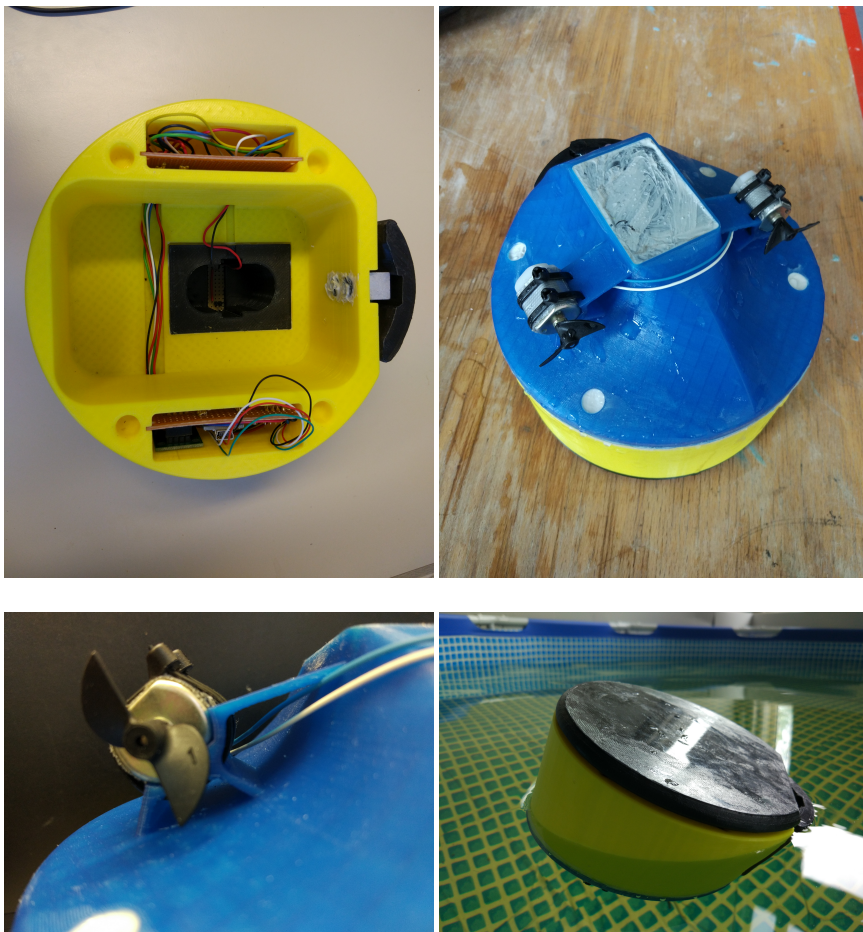
The tugboat is split in two main parts - Hull and Body - along a horizontal plane. The Hull will hold the motors, and will have a narrowing keel-like shape around the battery connector, with some semblance of a streamline design. This pushes the centre of buoyancy up and the centre of mass down, making the boat more stable in the water.

A third part is made to house the battery connector, as this must be inside the Hull, but have its opening towards the Body. Also, the design for the battery connector is not

finalized, and by making this a separate part, I retain the freedom to redesign this part without affecting the rest of the assembly.

The two final parts are a lid, to keep the electronics safe from water splashing, and a bumper, which can absorb the pushing force exerted by the tugboat, and direct it through the load cell. The lid has a flat top, to hold the Aruco marker, a requirement set by the computer vision software module.

The parts are assembled using 4 wood screws, silicon sealant, and 4 bolts to attach the load cell. The two motors are attached to the hull using 4 plastic strips and small pieces of tape to provide grip for the strips. This assembly method was quick and simple, and remained throughout manufacturing. Minion can be seen in Fig. ??.



**Figure 3.7:** Minion, physical prototype generation 1

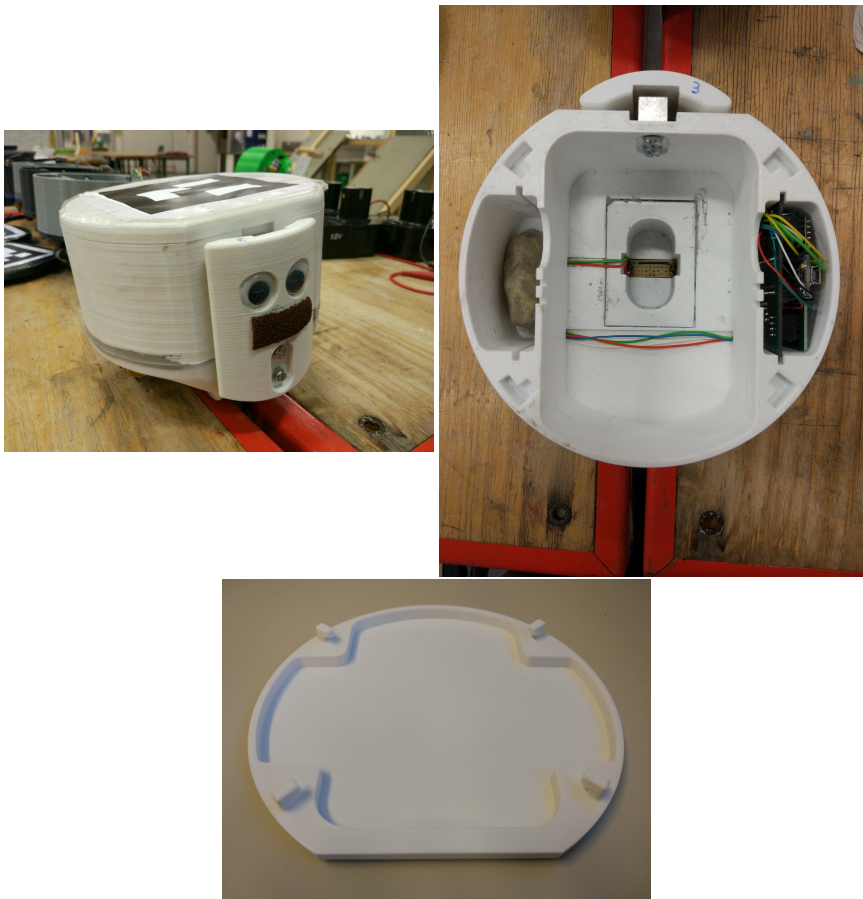
## Generation 2

The general design of Minion was satisfactory. The placement of the motors and the shape of the hull allowed for a good water flow through the propellers, and resulted in good thrust and turning capabilities. I was impressed with how well the printers managed to print the motor mountings, see Fig. ???. This was an experimental design, which would make for a very simple assembly process, but which I was not confident that the printers would manage.

The weight to volume ratio was within acceptable limits the tugboat is low in the water, but not problematically so. However, the balance was not very good, as can be seen in Fig. ???. After the impressive performance on the motor mountings, I was confident enough to make the rear wall thinner, allowing me to move the battery further back. The motors were also moved slightly further back, resulting in a much better balance.

I suffered from CAD blindness during design of Minion. In the CAD model, a 5mm bolt fits perfectly through a 5mm hole. In the real world, this is not so, particularly with an unprecise manufacturing method such as low-end 3D printing. Extra space was added around bolts (both around the heads and shanks), circuit boards and load cell to account for this. Finally, the electronics compartments were expanded to fit circuit boards with removable components, and a groove was made in the wall between each electronics compartment and the battery, to make the battery easier to grip.





**Figure 3.8:** Physical prototype generation 2.

With all of these changes, the 2<sup>nd</sup> tugboat prototype proved good enough for small-scale mass production, with only one small concern. When accelerating, the tugboat rocks in the water. This is because of the compact design, which I want to retain to comply with the size criterion set in Table 3.1. I will try to fix this in software, but as a precaution, I make four mounting holes for a keel on the bottom of the battery housing. The final result can be seen in Fig. 3.8

### 3.4 Manufacturing and assembly

3D printing turned out to be a good manufacturing process for this project, as I was able to manufacture parts for 6 tugs without much active involvement. As anticipated, several prints ended up failing, although this was in about half the cases caused by two power outages, and not machine failure. Nevertheless, my decision to split manufacturing into smaller print jobs was well justified, as this reduced the total amount of wasted print time.

Beyond this, only the motor assembly has room for reduction in part number. The motors are held in place by two zip ties and a piece of duct tape (to provide grip for the zip ties), as seen in Fig. ???. The number of parts could have been reduced from three to zero if the motors were fastened using an interference fit, but this would take considerable more time in the design phase, as the printers' tolerances would have to be investigated further. For total lead time, my simple solution works well.

Total active production time for each tugboat is approximately three hours, which is far less than the upper limit of one day. I was able to produce 6 functional tugboats early in project week 13, which is within the limit set at the start of the semester.

## 3.5 Software

The software represents the largest part of the project in terms of workload. A large number of sensors, actuators and computing units must interact in a real-time system, which brings a host of problems. It is also the part of the project with the most uncertainty. This is partly because this is the field with which I have the least experience, but mostly because it interfaces directly with the user input, which is the control algorithm.

The software is essentially a large control loop, with all the auxiliary functionalities required to operate on the test setup. These include a manager module to handle overall decision making, hardware drivers to gather and process sensor data and to operate the tugboats, and a communication structure to bring it all together.

The practise of breaking a large software project into smaller, self-sustaining modules is considered best-practise in software development. Smaller modules are easier to build, test and maintain. In this project, modularity is especially important because the part of the software that implements the ship control algorithm could be changed drastically from algorithm to algorithm, even to the point of changing programming environment.

I let *Design For Modularity* Eastman (2012) be the guiding design principle on the software side of the project. When designing for modularity, it is important to have simple, clear interfaces between the different modules. A good module does only one thing, and does this well.

### 3.5.1 Controller design

Three distinct control problems were presented in the introduction: Waypoint tracking, contact control and ship control. A controller must be designed for the first two, and the interface for the third must be established.

A simple 2-dimensional line-of-sight guidance system is chosen for waypoint tracking Fossen (2002), in which the heading controller and the speed-controller are decoupled. This should be sufficient for our purposes, as there is no inherent value in tracking a specific path. The goal is simply to get the tugboat to the endpoint in a reasonable time.

The setpoint for the heading controller is directly towards the next waypoint, while the setpoint for the speed controller is given by the path-planning, as part of the waypoint message. Both controllers are implemented as PID controllers, and will be tuned in the field, using all or some of the controller parameters. The speed controller has a simple



cut-off mechanism which sets the output to zero if the heading error is outside a given error of tolerance.

The setpoints for the contact controller are tugboat orientation and pushing force. The pushing force controller is a typical reference tracking problem, which can be solved using a simple open loop feedforward controller, or a combination of feedforward and closed loop feedback control (Balchen et al., 1971), using the load cell for feedback. Feedforward is especially useful for this application because the relationship between the input and the output is not a differential equation. Thus, a normal reference error feedback controller would have had to rely mostly on its integrator effect, which would cause large overshoots. The controller is implemented as the latter of the two discussed options, with the option of disabling the feedback controller to achieve the former. The orientation controller is a normal PID feedback controller, like the one used for waypoint tracking.

The ship controller needs information about the tugboats under its control, which it receives from a Tugboat Manager node. The Tugboat Manager filters information from the system, and pass along only the information that is relevant to the ship controller. The ship controller also need to know its own position, and its next waypoint. The output from the ship controller is setpoints for the contact controllers for each active tugboat.

One thing that must be clarified is that, while all ship control happens in a single module on the main computer, artificial barriers can be placed between the parts of the control algorithm that deals with each tugboat. This way, I can simulate local decision making and control algorithms where each tugboat has limited or no information about other tugboats. This is necessary to keep the tugboat drones in the test-setup as simple as possible, whereas in a real-world scenario, the tugboats would be equipped with more advanced sensors and processing power for navigation and collision avoidance.

Based on this, a preliminary diagram for the program modules and their interactions was made, which is presented in Fig. 3.9. From this plan, I can start building self-contained working modules, in spirit with the Agile methodology, but first I need to define a communication protocol.

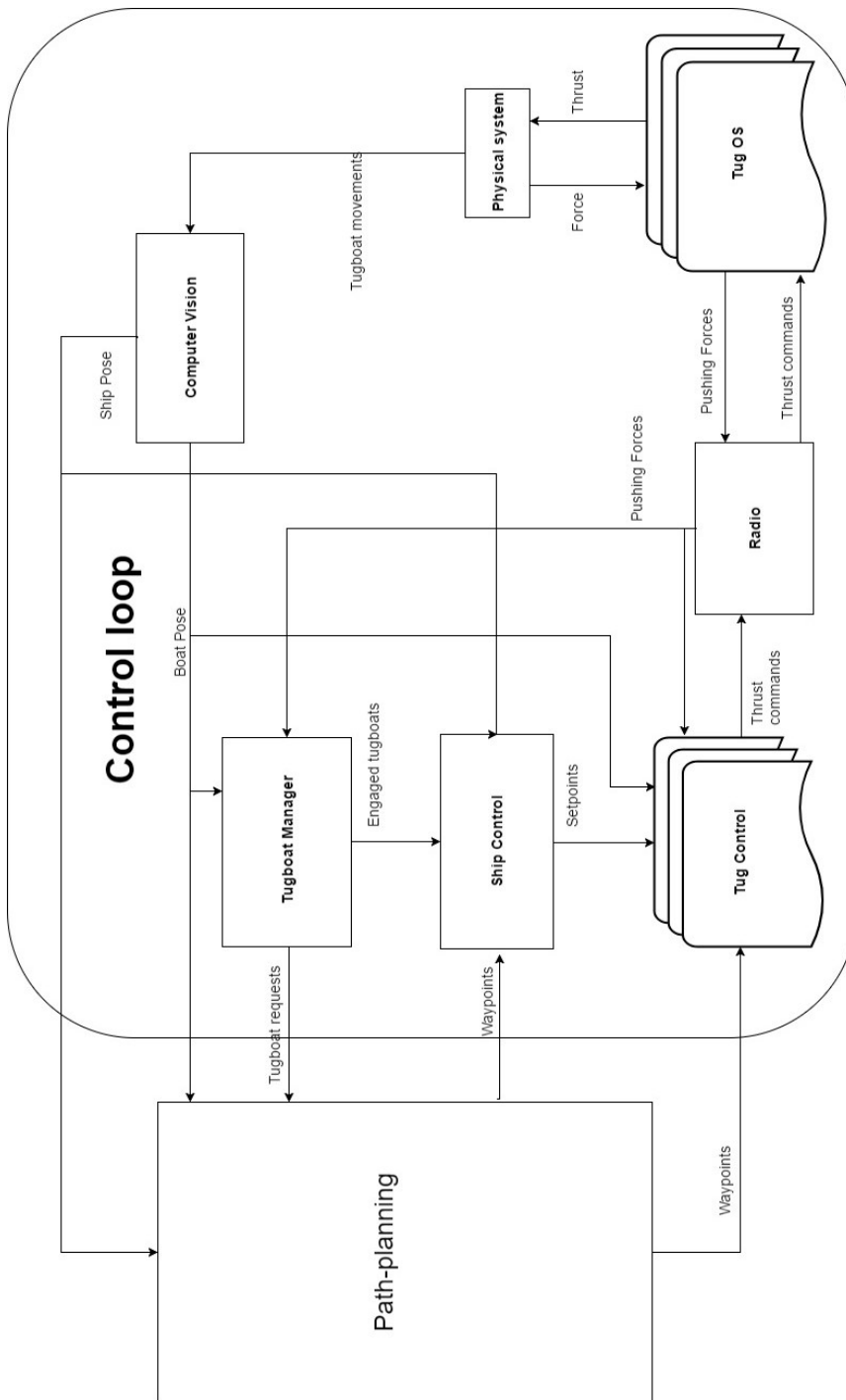


Figure 3.9: Initial software design

### 3.5.2 ROS

By recommendation from more experienced programmers, I (we) chose the *Robot Operating System* (ROS) as the communication framework for my program. In ROS, each module is a *node*, which communicates with other nodes through *messages* (data packets) published on *topics* (channels), and *services* (function calls). This communication protocol is excellent for one-to-many, many-to-one and many-to-many communication, which accommodates simple expansion of tugboat swarm. This is all facilitated by a *ROS Master*, which automatically handles multi-threaded scheduling and module communication.

ROS makes it easy to define clear interfaces between my different nodes, as well as combining my program with Cox path-planning software, which means we can work on our systems separately, and bring them together towards the end of the project. Changing between different ship control algorithms is simply done by shutting down one ship control node and starting up another. In addition to working in C++ and Python, ROS is available for MATLAB, which has a lot of very powerful tools for advanced control theory. Ros can also interface with Arduino microcontrollers, providing a simple link between my main computer and the world of microcontrollers.

Further, ROS is widely used in the industry and in research, which, in addition to providing a quality stamp, means that there is a lot of good documentation available online, and it is very useful to learn.

### 3.5.3 Final program flow design

The program is written in C++, as the Aruco sub-library in OpenCV is written in this language, and sets the precedence for the rest of the program. The only exception is the Ship Control node, which can be written in a number of languages better suited for control applications. ROS supports Python, with the NumPy package, and MATLAB, which both excel at linear algebra. The ship control node can even be modelled as a block diagram in Simulink. But for the proof of concept algorithm, C++ is used. The final program structure and communication flow is presented in [fig]. The Path Planning module is treated as a black box with a pre-defined interface. All the nodes are listed and briefly described below. All custom message types, services and nodes are presented in Appendix B.

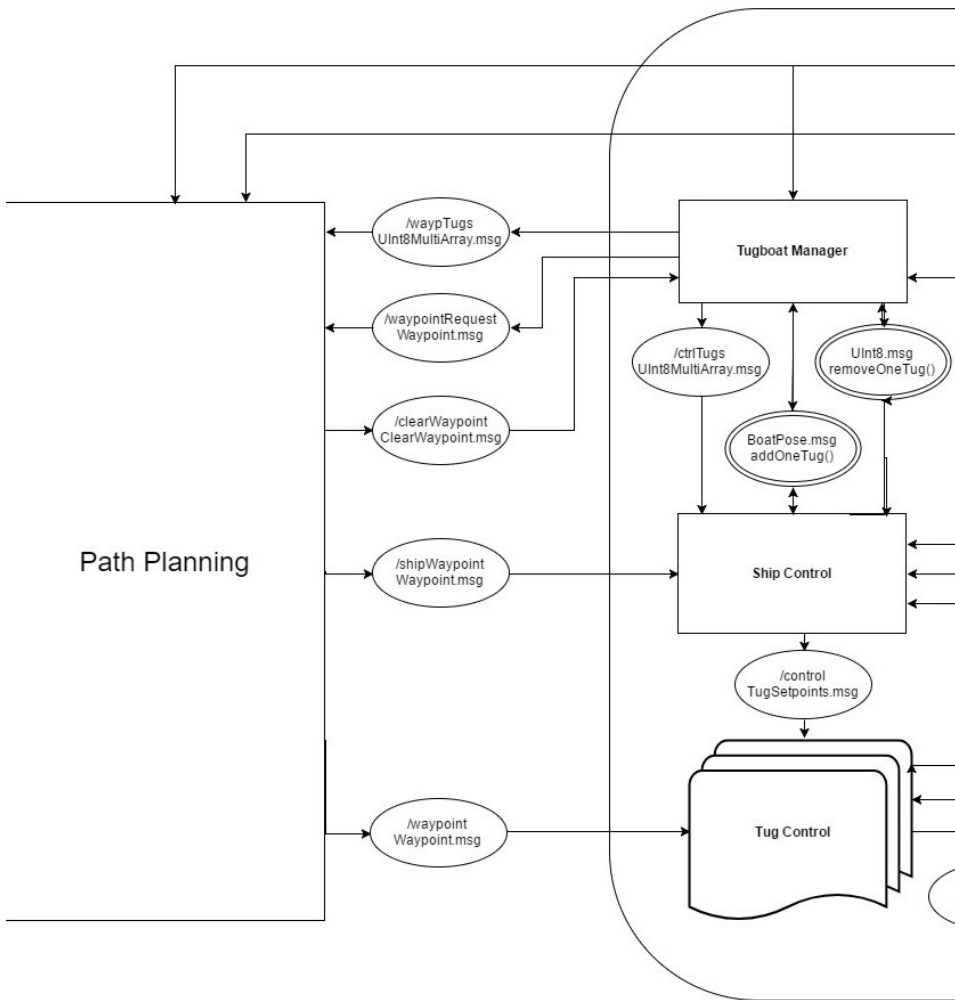
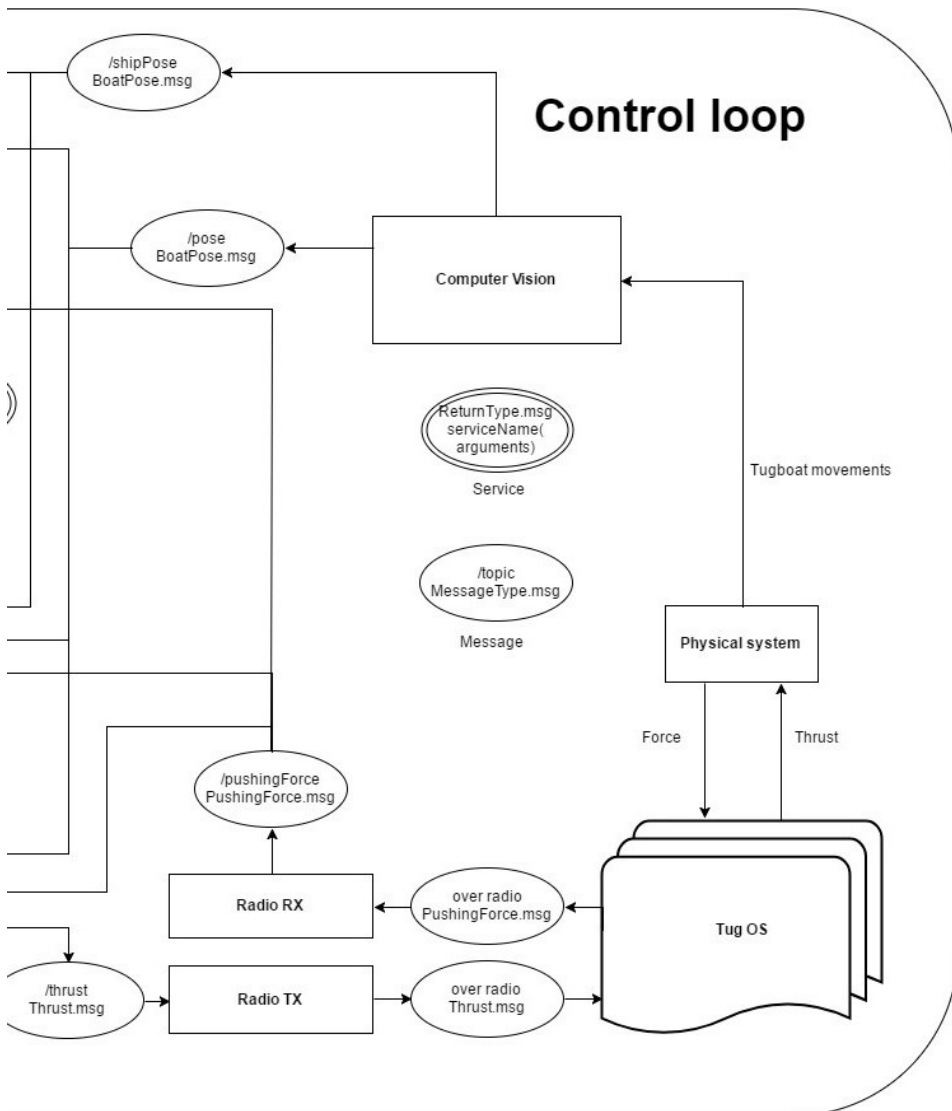
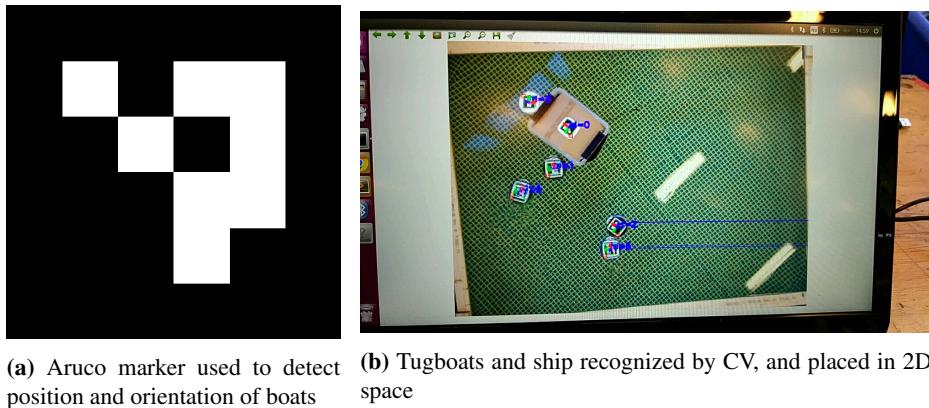


Figure 3.10: Final program flow design



## Computer Vision

Computer vision is used to find the pose (position and orientation) of the ship and each tugboat, which is the starting point for each iteration of the control loop, in place of a GPS-based navigation system. The OpenCV library is used for this, but rather than detecting coloured blobs, as described in the project thesis, Aruco markers are used. The Aruco module in OpenCV already has the same functionality I intended to achieve using blob



**Figure 3.11:** Computer Vision is used to find the pose of the boats

detection, and requires far less customization. Each boat has an Aruco marker on its top, with a unique pattern and a fixed size, see Fig. 3.11a.

The program can find and distinguish these markers in an image provided by a web-camera, and, when the camera is properly calibrated, the program can calculate the position and orientation of each marker in 3D space, see Fig. 3.11b.

### Tugboat Manager

The tugboat manager node is the master node, equivalent to the main function in a normal program. It must not be confused with the ROS Master, which is a behind-the-scenes communication and threading facilitator. Tugboat Manager keeps track of which tugboats are active in the test setup, which are in Waypoint mode and which are in Ship Control mode, and facilitates switches between the two.

A method for dynamic tugboat employment and extraction is implemented in the Tugboat Manager, but was never fully realized. The principle is based on *distress* calls. Each tugboat sends a positive distress call if its output thrust is above an upper threshold (i.e. working hard), or a negative distress call if its output is below lower threshold (i.e. hardly working). These calls are accumulated by the Tugboat Manager ( $stresslevel = \sum(positivecalls) - \sum(negativecalls)$ ), which can dispatch a tugboat to help out if the stress level is high, or extract a tugboat if it is low.

### Tugboat Manager

The Ship Control node will be different for each control algorithm, but the interface with the rest of the program, the structure of the code and the private function calls are the same for each implementation. The difference lies in the internals of three of the functions. These are *computeControlOutputs()*, which contains the actual control algorithm, and the services *addOneTug()* and *removeOneTug()*, which decide where along the hull of the ship a new tugboat should be added, and which tugboat should be released from the control

when the number of tugboats required to control the ship is reduced. This logic is tied to the control algorithm, and will therefore differ between implementations.

In the Ship Control node, the pose of the tugboats is given in ship coordinates. This makes it easier to formulate control algorithms, as these will be independent of ship position and orientation. The way ROS works forces a change in how Ship Control gets its information. ROS nodes run through a loop at a constant rate, which can be different from node to node. Ship Control depends on fresh pose and pushing force measurements, which makes it desirable to get these directly from the source, rather than through another module. This lets me use a slow rate for Tugboat manager, saving some computing power.

### **Tugboat Controller**

A tugboats behaviour must be different when running on waypoints and when it is part of a larger feedback control system. The tugboat controller is written as a final state machine, with two states (Waypoint mode and Ship Control mode) and two quasi states (Startup and Timeout). The states are triggered by any controller input message from their respective topics, which activates the relevant controllers. Timeout is triggered if no such message is received over a given period, and Startup is a passive state the tugboat assumes on startup, before any controller input messages are received. Both Startup and Timeout outputs all-zero thrust commands.

### **Radio RX and Radio TX**

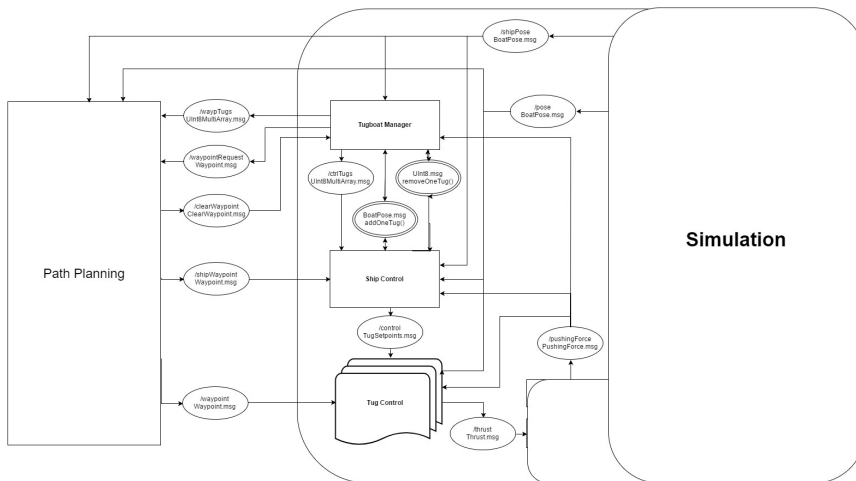
These two nodes run on two separate Arduino Nano boards, and provide the link between the main computer and the tugboats. Radio TX picks up thrust command messages published by the different Tugboat Controllers, and broadcasts them on the TX frequency. Radio RX collects Pushing Force messages broadcast on the RX frequency, and publishes them on the ROS network.

### **TugOS**

TugOS is the program that runs on each tugboat, on the on-board Arduino Nano. In each loop, it receives Thrust messages on the TX frequency until one matches the tugboats ID, and adjusts the motor thrusts accordingly. The maximum thrust change for each loop is limited, to make acceleration more smooth. The tugboat then takes a measurement of the pushing force, and broadcasts this on the RX frequency. If no applicable thrust message is received within a given wait period, currently 200 milliseconds, the motors are shut down.

### **Simulation**

The modular software design allows me to simply substitute the parts of the software that deal with the physical world with a simulation, see Fig. 3.12. A good simulation makes testing and debugging of algorithms quicker, with no setup- and maintenance time required for the physical setup. Unfortunately, due to time constraints, the simulation node was not finished.



**Figure 3.12:** The Simulation node neatly replaces all the nodes which interface with the real world, which means the rest of the program can run normally. The figure is only used to illustrate this point; details of the diagram are the same as in Fig. 3.10

### Simulation

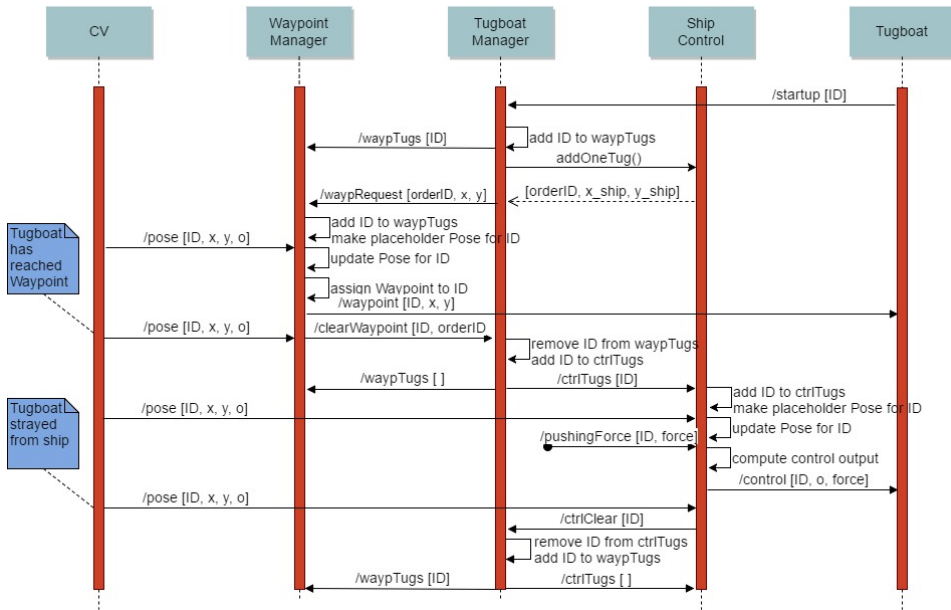
Systems that work well in a predictable simulated environment, fail when exposed to the uncertainties of the real world. Messages are lost, tugboats miss their target, or lose radio communication. Tugboats may disappear from view, or tugboats that do not exist are detected by the CV module. ROS handles scheduling (which thread in the program runs at any given time) but other real-time problems must be handled manually. The setup will inevitably experience packet loss, or situations where a packet is expected, but is never sent. This could happen when the computer vision module fails to detect a particular Aruco marker, or if a tugboat is outside the image frame.

To make sure tugboats do not go speeding off out of control when this happens, timing checks are put at two locations. The first is in the Tugboat Controller module. If no new BoatPose message is received over a period of 1000 milliseconds, the controller is shut down, outputting a Thrust message with zero on thrust and turn. When a new pose message is received, the controller is reactivated, but the controller parameters are reset, to avoid unnatural spikes in integration and differentiation effects. The second is in TugOS, locally on each tugboat. If no new Thrust message is received over a period of 200 milliseconds, the motors are shut down gradually, to avoid jolt.

One of the most challenging real-time problems is managing the tugboats as shared resources between the Waypoint module and the Ship Control module. The program must be designed in such a way that no tugboat is lost in transition, or being controlled by both modules at the same time, causing a race condition.

Even more important is to know what information you have at any given time, as an attempt to access information that does not exist will result in a segmentation fault, causing the program to abort. This could happen if, for example the Ship Control module tries to





**Figure 3.13:** Sequence diagram showing tugboat and waypoint handling between modules. Note that the Waypoint Manager in this sequence diagram is my placeholder module, which I made in order to test my own program, before implementing Cox path planning. The inner workings of this module will therefore differ from that used in the final product, but the interactions with other modules remain unchanged.

use a newly assigned tugboats position to calculate its control output, before the position of this tugboat is detected and stored in the memory.

To illustrate how this has been resolved, a sequence diagram, Fig. 3.13, is presented, showing how a tugboat is handed between the modules. The key element is that the official lists of which tugboats belong to which module, and which waypoints are requested from the Waypoint Manager by Ship Control, all resides with the Tugboat Manager. Every second, these lists are sent to the respective modules, which update their local versions accordingly. This redundancy ensures that no tugboats or waypoints are lost in transition, which is the main purpose of the Tugboat Manager module.

## 3.6 Combining the subsystems

By this point, all the different modules have been tested vigorously, but only in strictly controlled circumstances, in very limited scenarios. When combining all the modules into one product, I can finally see how they interact with each other, and how they react to a real-life setting. This represents a whole new design iteration, with new insights, ideas and requirements.

It also provides a measure of how well I have executed Concurrent Engineering. This is the phase when those intermodular problems CE strives to minimize, come to the surface

and cause problems, potentially resulting in massive rework. An overview of the major problems and bugs, and their resolution, is presented below, to show how well I did in design. The final test setup can be seen in Fig. 3.14.

### 3.6.1 Problems and bugs

Computer vision does not work as well as hoped in a changing-light environment. The program loses track of markers for periods of time, which cause a cascading series of problems. Estimates for translational and rotational velocity become unreliable, which affects the speed controller and the derivation effect in heading controller. This, in turn, gives poor waypoint tracking. This problem was alleviated by removing the clear tape used to waterproof the Aruco markers, which resulted in much less glare.

The camera was placed at a moderate distance above water, to further improve Aruco detection. This had the unfortunate side-effect of allowing the tugboats to move out of the picture, effectively vanishing from the setup. A wooden frame was built to contain the tugboats, and also to provide an anchor for the harbour elements used to demonstrate Cox path-planning algorithms.

Because of the unstable position control feedback, it became more important to have a self-stabilising tugboat design. Not in pitch, as I had anticipated, but in yaw. A very simple keel was made from a thin sheet of aluminium, see Fig. 3.15. The keel was attached to two of the four holes underneath each tugboat, which were put in the design for this exact eventuality. The keel helps dampen out spin, allowing the tugboat to rely less on the derivative effect in the heading controller, and aligns the tugboat with the direction of travel when moving at speed.

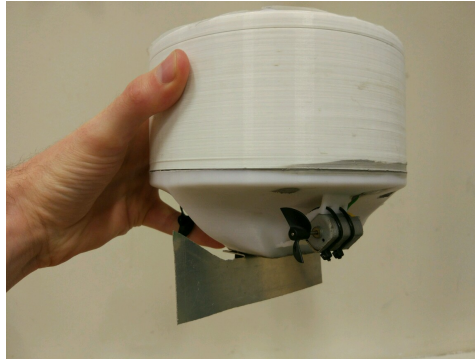
The load cell measurements were practically useless. Each load cell is individually calibrated to give the correct pushing force in Newtons, by setting the gain on the signal from the ADC and to set the tare weight. This is done in the setup function in TugOS, which runs immediately after start-up, when the battery is inserted into the tugboat.

The problem arises when the load cell is submerged in water, which changes its properties. This resulted in static offsets in the measurements of up to an order of magnitude larger than the measured range, which is devastating when using the measured property to run a feedback control loop. An attempted solution was to simply program the tugboats to reset their tare weight upon receiving the first Thrust command message, which happens when they are all in the water, not pushing anything. While this helped with the static offset, the load cells did not give reliable measurements, and were prone to drifting. This might be due to the influence of water, but at this point an investigation was not prioritized. Instead, the feedback-part of the pushing controllers were switched off for the remainder of the project.

These were all small changes, quickly resolved. No major design flaws forced me to greatly redesign parts of the setup, which was the goal from a project management viewpoint. All in all, bug fixes and design rework took a few days to sort out, rather than a few weeks, which in a 20-week project is pretty good.



**Figure 3.14:** Macro shot of the whole test setup.



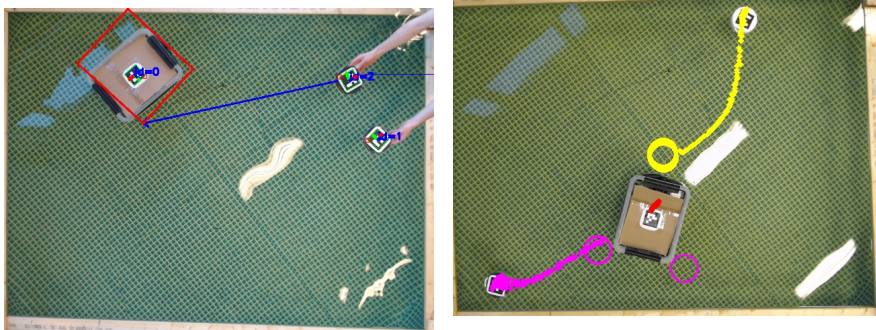
**Figure 3.15:** Retro-fitted keel mounted to pre-existing holes, to improve motion stability.

### 3.6.2 Visualization

ROS provides a simple tool for monitoring the messages that are passed between modules, in which the messages are printed live to a terminal window. This was sufficient information when developing my own program, but when Cox module was integrated with my system, it quickly became evident that a better user interface was needed. Mostly because Cox algorithms have dynamic goals with longer time horizons, which are harder to track, but also simply because she did not have the same familiarity with my system as me, and was therefore not able to extract the same information from the terminal windows.

Two important visualization effects were added to the video feed. The first was arrows pointing from each tugboat to its current waypoint. The second was a rectangle around the ship, to illustrate how the ship is modelled in the path-planning software. At the same time, functionality for recording the boats movement and their waypoints was added, which is useful for documenting the performance of the test setup. The two visualization outputs can be seen in Figs. 3.16a and 3.16b, respectively.

These effects proved invaluable when debugging Cox module. For example, the rectangle around the ship immediately revealed that the model used by the path-planning module was rotated the wrong way, as seen in Fig. 3.16a, which gave the tugboats trouble rounding the ships corners. This was due to the left-hand coordinate system produced by the CV module, which was different from the coordinate system Cox had used to test her algorithms.



(a) Live camera feed, with visualization of waypoint tracking (blue arrow). Note how the square marking the safe-zone around the ship is rotated the wrong way.

(b) Session recording feed for analyzing movement and waypoint tracking of both tugboats and ship. The circles indicate waypoints, and the uneven curves track the movement of each tugboat.

**Figure 3.16:** The two visualization channels produced by the setup. Each boat has a unique colour.



## Test setup FAT

In this chapter, the test setup will be evaluated based on how well it can perform each of the five steps mentioned in chapter 1.3, which comprise the FAT. To test these steps in a practical setting, a very simple ship control algorithm was devised. A rectangular box, roughly an order of magnitude more massive than a single tugboat, was used as a ship, as this is a simple shape to model.

The tugboat allocator in this algorithm places the first four tugboats midway along of each of the four sides of the ship. Subsequent tugboats are placed midway along the side which currently has the fewest tugboats. The controller outputs – orientation and pushing force setpoints for each tugboat – are respectively directly normal to the ships hull, and given by the following equation:

$$pushingForce_n = \max(F_{contact}, F_{max}\cos(o_{shipsetpoint} - o_n)), \quad (4.1)$$

where  $F_{contact}$  is the estimated force required to maintain contact with the ship,  $F_{max}$  is the maximum force the tugboats are allowed to deliver,  $o_{shipsetpoint}$  is the angle of the vector pointing towards the ships waypoint, and  $o_n$  is the orientation angle of the  $n^{th}$  tugboat.

### 4.1 Waypoint tracking

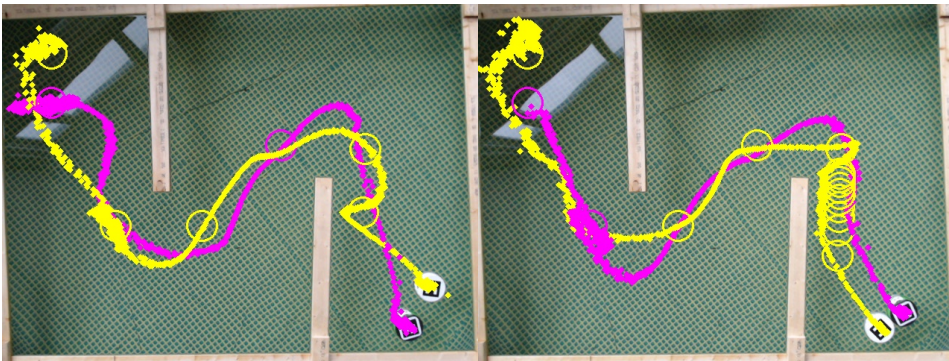
Figs.4.1 shows two tugboats following a path produced by Cox path-planning algorithm. The thick lines show the trajectories of the tugboats, and the circles indicate the intermediate and final waypoints which comprise the path. Each tugboat is represented by its own colour, but note that some of the waypoints are the same for both tugboats, and the indicator circle is overwritten by the last tugboat to be assigned a specific waypoint. The waypoints along the bottom right obstacle are generated when the yellow tugboat moves inside the safety-zone defined around the obstacle.

It can clearly be seen that the path between two consecutive waypoints is not a straight line. This demonstrates the drawbacks of using waypoint tracking, as opposed to trajectory



following, as discussed in Chapter 1.3. While there are no significant currents or other external forces in the test setup, the path of the tugboat is disturbed by internal factors. To simplify the design, the same controller parameters were used for all tugboats. But each tugboat, although identical in design, differ slightly from the others. Factors which affect a tugboats dynamic properties are the machine and the settings used when 3D printing the parts, small differences in the motors, and most notably the weight distribution of the battery, which varies from one battery to the next, and its charge state. The pose estimated by the CV module is not perfect (but within the required range), nor perfectly regular, further disturbing the controller.

However, as explained in Chap. 3.5.1, the goal is simply to get the tugboat to a given location within a reasonable time, so this waypoint tracking should be more than sufficient for our purposes. The tugboats' movements are indeed similar to that of a duck swimming leisurely in a pond.



**Figure 4.1:** Two tugboats tracking a path from one corner of the setup to the other, two different runs.



**Figure 4.2:** By using a smartphone to scan this QR code, the reader will taken to a video of a similar waypoint-tracking run on the test setup.



## 4.2 Contact control

While the feedback from the load cell did not work as anticipated, the simple feedforward controller performed admirably in its stead. This give a less precise output than a feedback controller would, but is nevertheless sufficient for the simple ship controller used in the FAT. It is a direct violation of product requirement 2.3 in Table 3.1, but since the requirements were designed to make sure the test setup pass the FAT, this is still technically a success.

On a side note, the round shape of the tugboats makes the controller system asymptotically stable, as any positive thrust (Which is guaranteed by the controller) will work to align the tugboat normal to the hull of the ship.

## 4.3 Initiating contact with ship

Reliably initiating contact with the ship was one of the most challenging parts of the project, and remains one of the weakest points in the test setup. With current controller setup, there is a period in this transition when the tugboat is between states – it is running on the contact controller, but does not have contact with the ship. During this period, the tugboat must retain the characteristics of the heading controller, which is tuned to be far less aggressive than the contact controller could potentially be.

This way, the contact controller is limited by this stage, so a good idea would be to make a third control state, which is used from the moment the tugboat is released from the Waypoint Manager until the it has achieved steady contact with the ship. This controller will be on the same format as contact controller, but tuned similarly to waypoint controller.

Further, the notion of a waypoint is lost when transitioning to contact control, as the only special concerned in this mode is with achieving and maintaining contact with the ship. This could also be addressed in third controller mode by steering the tugboat towards the point requested by the Ship Control module. This way, tugboats could more easily be placed in particular arrangements, thus paving the way for a range of more formation dependent control algorithms.

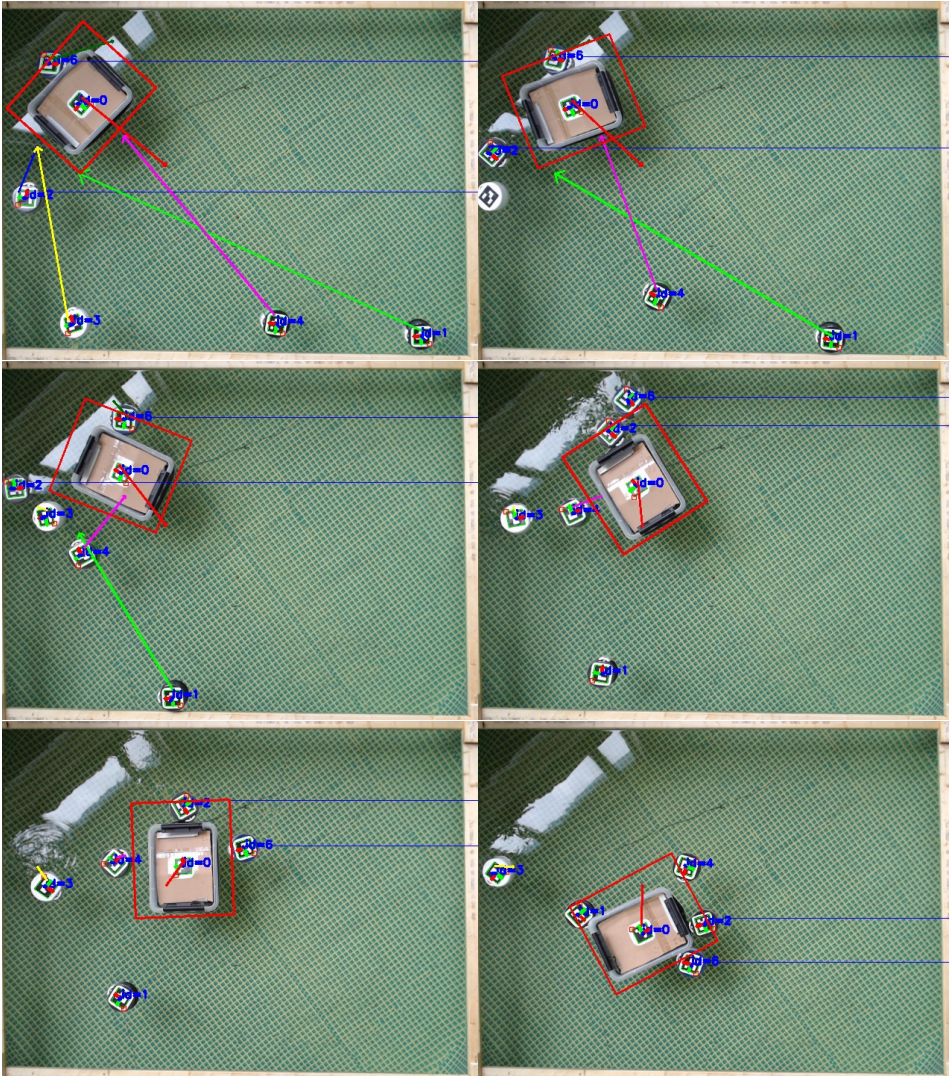
All this aside, the test setup *is* able to reliably initiate and maintain contact with the ship, as can be clearly seen in Fig. 4.3.

## 4.4 Surrounding the ship

The challenge with this step lay in timing. If one or two tugboats reach the ship and start pushing, they may push the ship away from the other tugboats, or make it spin too fast for the other tugboats to initiate contact. The solution is to have the tugboats wait at their target position until enough are ready for the group to gain some control over the ship.

This timing was handled in Cox' module, as that module is responsible for checking if tugboats have reached their final waypoint, before releasing sending them back to the Ship Control module via the Tugboat Manager. This step would not have been possible without Cox' path-planning algorithms, which helps the tugboats move around the ship to reach the far side.

However, this is the step that requires the highest precision and reliability from the waypoint controller, both to get to, and remain at the final waypoint. An image sequence showing a successful attempt at this step, which includes verification of steps 1 and 2, is depicted in Fig. 4.3. A video showing a similar situation can be seen by scanning the QR code in Fig. 4.4



**Figure 4.3:** Tugboats surrounding ship in order to gain control. Time line progresses from upper left to lower right.



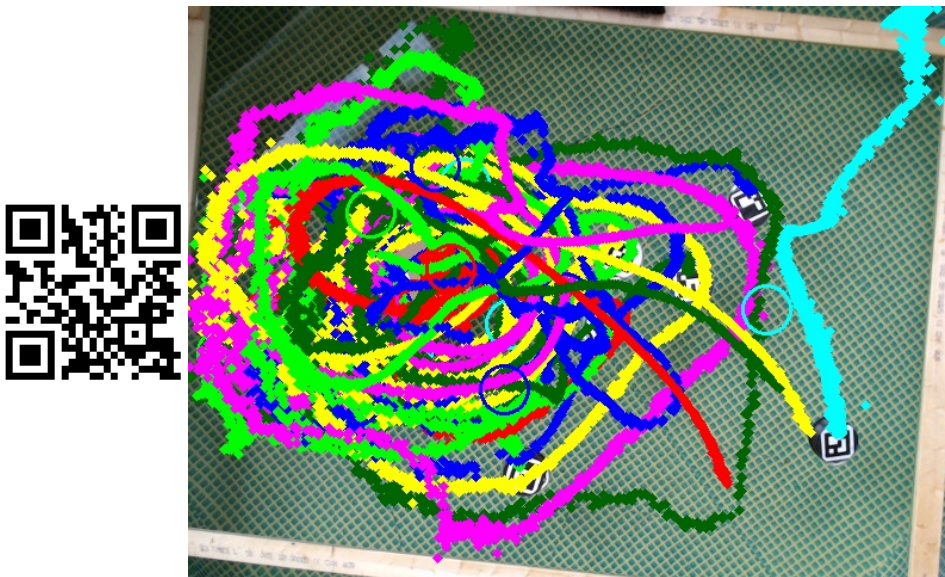
**Figure 4.4:** By using a smartphone to scan this QR code, the reader will taken to a video of tugboats surrounding the ship and achieve control.

## 4.5 Controlling the ship

The ship control algorithm described in the beginning of this chapter was simplistic bordering on the ridiculous, but given the right tugboat arrangement, it actually works. Fig. 4.5 depicts the path of all boats during a session in which the ship was controlled to within the vicinity of its waypoint for 5 consecutive minutes. This was far beyond the author's expectations.

We can see from experimental results that, given the right tugboat arrangement, the control algorithm is marginally stable. This means that the distance from the ship to the waypoint will be bound by some upper threshold.

I consider this a huge success for the project, and a solidly passed FAT.



**Figure 4.5:** By using a smartphone to scan the QR code on the left, the reader will taken to a video of 5 tugboats controlling the position of the ship to a contained area around the ship's waypoint for 5 consecutive minutes. The waypoint is indicated by the red arrow going from the ship to a static point in the middle of the left half of the videofeed. The image to the right is the path taken by all boats during the video. The ship's path is marked by a red curve. The teal tugboat was broken down during the session, which can be seen in the video.

# Project evaluation and further work

This project is not just about making a product that can pass a FAT. Primarily, it is a product development master's thesis, with the aim of using PD tools and methods to solve a PD challenge, and to document this process. It is also a research project for Kongsberg Maritime on autonomous tugboats, which should bring some value to the company, and hopefully motivate continued research.

## 5.1 Work methods

The framework for product development described in chapter 2, was crucial for the success of the project. In particular, the Agile methodology, which allowed me to finish a project of this magnitude in so little time, by keeping a rapid pace, focusing on producing functional prototypes early and continuously. I spent most of my time directly working towards finishing the test setup, adding value to the project, and less time on necessary waste and pure waste (Welo and Ringen, 2016).

The different DFX tools I used in the project helped further speed things up, by having a simple framework in which to ground my design decisions. My adaptation of Concurrent Engineering ensured that all aspects of the project moved along at a steady pace, as I was always aware of which aspects were falling behind, or contained the most uncertainties. It also helped prevent major rework, by giving me a grasp of the co-dependencies of the interacting aspects of the project. Im not claiming there are no flaws in my design, but none of the flaws were crippling, or required major rework.

I did not follow any one methodology rigidly, but rather extracted the most relevant tools and ideas from different methods, to tailor-make a work methodology for me, in this project. This worked well because I worked mostly alone, and had control over every aspect of the project. The same would not apply had I worked in a larger team, where other people relied on me to be predictable, and a shared, solid foundation for making decisions would be necessary.

## 5.2 Test setup

The test setup has been the main focus of this thesis, and represents the measurable results of my project. We were able to demonstrate a successful FAT in front of Espen Strange from Kongsberg Maritime, who showed great interest in the project. The demonstration was very much a joint effort between Cox and myself, but in this chapter, I will focus on my parts of the setup.

### 5.2.1 Evaluation

As has been stated repeatedly throughout this thesis, this is a huge, complex project. As two hard-working master students, it has been all we can do to get something up and running, and to be able to demonstrate a proof of concept ship control algorithm on the setup. Accordingly, the test setup is far from perfect. The Aruco markers are sometimes not detected properly by the CV module, the radio communication is prone to packet loss, and the tugboats have a hard time going in a straight line, just to mention a few things.

But perfect was never the goal. It is important to emphasize that this is an early prototype in what will hopefully prove to be a large development project. Nevertheless, the test setup passed all points in the FAT described in the introduction of this thesis, although the ship control algorithm implemented was not the most stable.

And the modular design of the control software makes the setup highly adaptable, not only suited for the many possible approaches which fit in the current guidance, navigation and control based configuration, but these modules could simply be replaced by systems relying wholly on swarm intelligence or machine learning.

The setup represents a simplistic implementation of autonomous tugboat operation, and it is this simplicity that gives it value as a research platform. It is exceedingly cheap to produce and operate, and the tugboats are simple to model and control. And yet, the feedback from the setup is very realistic. Problems faced in the setup are the same as the problems faced in the real world, such as dealing with interference from other tugboats, or keeping the batteries topped up.

### 5.2.2 Further work

The first thing that comes to mind for further work, is to upgrade tugboats with more sensors and processing power, to implement inertial navigation with Kalman filtering, dynamic collision avoidance and all the other features associated with autonomous vessels. But that would defeat the vision of the setup, which is to keep it simple to use, cheap and adaptable, and to provide feedback on aspects directly related to SCAT.

The focus for further work should be features that enhance these properties, such as a good graphical user interface, in which users can change the number and behaviour of tugboats without having to manually edit a bash file. The GUI can provide better visual representations of what the tugboats are doing, like what I did with the waypoint arrows. Interesting features are motor thrust and pushing force of each tugboat, translational and rotational acceleration of the ship, amongst others. This would not only make the setup easier to use, but also increase the learning outcome from every session.



Features that would make the test setup more reliable, such as better wireless communication, better batteries and power management, particularly a system for automated recharging when a tugboat is running low on power. Better hydrodynamic stability would also help, as would an investigation into why the load cell did not work as expected, and a solution to get proper pushing feedback. In short, a hardware overhaul focused on reliability, rather than pushing lead time to the absolute minimum, would be useful.

The control loop could be improved by adding a more sophisticated filter to the measurement feedback, and make a more rigid framework for concurrency in the controller. A third control state could be applied to the tugboat controller, to handle the transition between waypoint tracking and contact control, as discussed in chapter 4.3.

Again, the modular design proves valuable, as any one feature can be upgraded without greatly affect the rest of the system.

## 5.3 Ship control

The focus of this project has been to get to the point where the tugboats can assume control of the ship, but I have barely scratched the surface of actually manoeuvring the ship.

Implementing and evaluating different ship control algorithms was originally part of the problem definition for my master's thesis, but was discarded due to time constraints. However, some thought has been put into this problem throughout the year, and I have come to certain insights during my work. I want to take this opportunity to discuss these thoughts, mostly for the benefit of anyone who will continue this project.

In the conclusion of my pre-master's thesis, one of my main concerns are with implementing and evaluating the control algorithms discussed in the literature review. While this is still of interest, this is only a small part of what the test setup is intended to do. In fact, I want to dispense with the term test setup, which has been used throughout this thesis for the sake of consistency. Instead, I want to call it a research platform, because this is where its true value lies.

Without significantly altering the platform, a whole spectre of aspects of SCAT can be studied, as the user is free to design the internal mechanics of the ship control node in any way. The following list contains aspects the platform is suited to research, and which I think are interesting in further development of SCAT.

- Different control algorithms
- Different AI motors for ship control; Swarm intelligence, machine learning.
- The number and size of the tugboats, relative to that of the ship.
- The effects of different tugboat arrangements on different control algorithms.
- Communication schemes; Top-down, tug-to-tug, minimal communication.
- Automatic deployment and extraction of tugboats.
- Problems related to communication failure in different parts of the loop, or faulty tugboats.

- How tugboats interact with, or are disturbed by each other.



# Chapter 6

## Conclusion

This project has been very interesting, both in terms of the work I have carried out throughout the year, and in terms of the greater vision. I strongly believe autonomy will shape the future, and being able to contribute to this development is one of my biggest motivations.

The collaboration with Cox has been great, as, with our complementary skill-sets, we had exactly the right qualifications for this project. We have learned a lot from each other, and provided support and motivation.

The performance of the test setup was, I think, as good as can be expected, with a mostly successful FAT. I am confident that the setup will be able to perform different (robust) ship control algorithms. I am especially satisfied with the modular design of the setup, which facilitates further development of the setup, in the direction best suited to whoever carries the project forward.

We were fortunate enough to demonstrate our work for Espen Strange at Kongsberg Maritime, who showed great interest in the project. We were proud of the work we presented, and it is nice to know that our work is appreciated. I hope, and believe, that the project contribute to the development of fully functional autonomous tugboats.

While the scope of the project forced me to take many shortcuts in technical implementation, often at the expense of performance and technical depth, the learning outcome in terms of product development management has been tremendous. Having acted as mechanical, electrical, control and software engineer, manufacturer and product manager in a single project, I have gained a unique perspective on the work-flow and interaction between these fields. I consider this experience invaluable in a world where the fields of engineering are more interwoven than ever.



# Bibliography

- Balchen, J. G., Fjeld, M., Solheim, O. A., 1971. *Reguleringsteknikk*. Vol. 5. Tapir.
- Boothroyd, G., 1994. Product design for manufacture and assembly. *Computer-Aided Design* 26 (7), 505–520.
- Eastman, C. M., 2012. *Design for X: concurrent engineering imperatives*. Springer Science & Business Media.
- Fossen, T. I., 2002. *Marine control systems: guidance, navigation and control of ships, rigs and underwater vehicles*. Tapir.
- Fowler, M., Highsmith, J., 2001. The agile manifesto. *Software Development* 9 (8), 28–35.
- Hirota, T., Ikujiro, N., 1986. The new new product development game. *Harvard Business Review* 64 (1), 137–146.
- Houde, S., Hill, C., 1997. What do prototypes prototype. *Handbook of human-computer interaction* 2, 367–381.
- Larman, C., Basili, V. R., 2003. Iterative and incremental developments. a brief history. *Computer* 36 (6), 47–56.
- Prasad, B., 1996. *Concurrent engineering fundamentals*. Vol. 1. Prentice Hall Englewood Cliffs, NJ.
- Steinert, M., Leifer, L. J., 2012. 'finding one's way': Re-discovering a hunter-gatherer model based on wayfaring. *International Journal of Engineering Education* 28 (2), 251.
- Ulrich, K. T., 2003. *Product design and development*. Tata McGraw-Hill Education.
- Welo, T., Ringen, G., 2016. Beyond waste elimination: Assessing lean practices in product development. *Procedia CIRP* 50, 179–185.
- Xue, D., Yadav, S., Norrie, D. H., 1999. Knowledge base and database representation for intelligent concurrent design. *Computer-Aided Design* 31 (2), 131–145.

---

---

---

# Appendices

---

---

# Appendix A

## Test setup source code

```
//CV.cpp

#include "ros/ros.h"
#include "tugboat_control/BoatPose.h"
#include "tugboat_control/Waypoint.h"
#include "tugboat_control/TugSetpoints.h"
#include <std_msgs/Float64MultiArray.h>

#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/aruco.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>

#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <math.h>
#include <time.h>

/*<-- Remove/add 1st / to toggle - fix this to use input
#define CAM 0 //Built-in camera - or not?
*/
#define CAM 1 //extern camera
/*<*/
#define OUTPUTMODE true
#define SHIP_ID 0
```

---

```

#define X_OFFSET 1.5
#define Y_OFFSET 1

#define markerLength 0.094 //m

std::vector<tugboat_control::Waypoint> waypoints(7);
std::vector<bool> inWaypMode(7);
cv::Mat videoImg, boatPosImg;
std::vector<cv::Scalar> colour(7);
std_msgs::Float64MultiArray shipFrame;
int framesSinceLastDrawMsg = 0;

static bool readCameraParameters(cv::Mat &camMatrix, cv::Mat
&distCoeffs) {
    std::string filename;
    if(CAM == 0){
        filename = "/home/sondre/catkin_ws/src/tugboat_control/
cameraParametersUSB";
    }
    else {
        filename = "/home/sondre/catkin_ws/src/tugboat_control/
cameraParameters";
    }
    cv::FileStorage fs(filename, cv::FileStorage::READ);
    if(!fs.isOpened())
        return false;
    fs["camera_matrix"] >> camMatrix;
    fs["distortion_coefficients"] >> distCoeffs;
    return true;
}

double rotationMatrixToEulerAngles2D(cv::Mat *R)
{
    double sy = sqrt(R->at<double>(0,0) * R->at<double>(0,0)
+ R->at<double>(1,0) * R->at<double>(1,0) );
    bool singular = sy < 1e-6;
    double o;

    if (!singular) {
        o = atan2(R->at<double>(1,0), R->at<double>(0,0));
    }
    else {
        o = 0;
    }
}

```

---



---

```

        return o;
    }

    void waypCallback(const tugboat_control::Waypoint::ConstPtr&
        waypIn)
    {
        waypoints[(int)waypIn->ID] = *waypIn;
        inWaypMode[(int)waypIn->ID] = true;
    }

    void shipWaypCallback(const tugboat_control::Waypoint::
        ConstPtr& wayp_in)
    {
        waypoints[0] = *wayp_in;
        inWaypMode[(int)wayp_in->ID] = true;
    }

    void ctrlCallback(const tugboat_control::TugSetpoints::
        ConstPtr& ctrl_in)
    {
        inWaypMode[(int)ctrl_in->ID] = false;
    }

    void drawShipCallback(const std_msgs::Float64MultiArray::
        ConstPtr &msg)
    {
        shipFrame = *msg;
        framesSinceLastDrawMsg = 0;
    }

    void drawShip()
    {
        if(shipFrame.data.size() != 8 || framesSinceLastDrawMsg
            > 10)
        {
            return;
        }
        framesSinceLastDrawMsg++;
        std::vector<cv::Point> pts;
        int i = 0;
        while ( i < shipFrame.data.size()-1)
        {
            pts.push_back(cv::Point(shipFrame.data[i],
                shipFrame.data[i+1]));
            i+=2;
        }
    }

```

---

---

```

    }
    int npts = cv::Mat(pts).rows;
    const cv::Point *pts_ptr = (const cv::Point*)
    cv::Mat(pts).data;
    polylines(videoImg, &pts_ptr, &npts, 1, true,
        colour[0], 2, 8, 0);
}

void saveImage(cv::Mat *img)
{
    std::cout << "Saving image\n";
    time_t rawtime;
    struct tm * timeinfo;
    char timeString [80];
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    strftime(timeString, 80, "%d.%m_%H:%M:%S", timeinfo);
    std::string part1 = "tugPosImg_";
    std::string jpg = ".jpg";
    std::string name = part1 + timeString + jpg;
    cv::imwrite(name, *img);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "CV");
    ros::NodeHandle n;
    ros::Publisher tug_pub = n.advertise<tugboat_control::
    BoatPose>("pose", 100);
    ros::Publisher ship_pub = n.advertise<tugboat_control::
    BoatPose>("shipPose", 1);

    ros::Subscriber shipWayp_sub = n.subscribe("shipWaypoint",
        1, shipWaypCallback);
    ros::Subscriber wayp_sub = n.subscribe("waypoint", 100,
        waypCallback);
    ros::Subscriber ctrl_sub = n.subscribe("control", 100,
        ctrlCallback);
    ros::Subscriber drawShip_sub = n.subscribe("drawShip",
        100, drawShipCallback);

    ros::Rate loop_rate(10);

    tugboat_control::BoatPose boat;

```

---

---

```

//CV init
cv::VideoCapture capWebcam(CAM);
cv::Mat cameraMatrix, distCoeffs, rotation3x3;

if (capWebcam.isOpened() == false) {
    std::cout << "error: Webcam not accessed
    successfully\n\n";
    return(0);
}

cv::Ptr<cv::aruco::Dictionary> dictionary =
cv::aruco::getPredefinedDictionary(cv::aruco::DICT_4X4_50);
std::vector< int > markerIds;
std::vector< std::vector< cv::Point2f > > markerCorners;
std::vector< cv::Vec3d > rvecs, tvecs;
cv::aruco::DetectorParameters parameters;

bool readOk = readCameraParameters(cameraMatrix,
    distCoeffs);
if(!readOk) {
    std::cout << "Invalid camera file\n";
    return 0;
}

std::cout << "CV node initialized successfully\n";

//This image is used to present boats' movement in thesis
capWebcam.read(boatPosImg);
//Each bot has its own colour
colour[0] = cv::Scalar( 0, 0, 255);
colour[1] = cv::Scalar( 0, 255, 0 );
colour[2] = cv::Scalar(255, 0, 0 );
colour[3] = cv::Scalar( 0, 255, 255);
colour[4] = cv::Scalar(255, 0, 255);
colour[5] = cv::Scalar(255, 255, 0 );
colour[6] = cv::Scalar( 0, 100, 0 );
int counter = 0;

while (ros::ok() && capWebcam.isOpened() )
{
    counter++;
    if(capWebcam.read(videoImg))
    { //Camera capture successful
        cv::aruco::detectMarkers(videoImg, dictionary,
            markerCorners, markerIds);
    }
}

```

---

---

```

if (markerIds.size() > 0)
{ //At least one aruco marker found
  cv::aruco::estimatePoseSingleMarkers(markerCorners,
    markerLength, cameraMatrix, distCoeffs, rvecs,
    tvecs);

  if(OUTPUTMODE)
  {
    cv::aruco::drawDetectedMarkers(videoImg,
      markerCorners, markerIds);
  }

  for(int i=0; i<markerIds.size(); i++)
  {
    if(OUTPUTMODE)
    { // Drawing functions
      cv::aruco::drawAxis(videoImg, cameraMatrix,
        distCoeffs,
        rvecs[i], tvecs[i], markerLength * 0.5f);

      cv::Point2d tugPt = cv::Point2d((tvecs[i][0]
        + X_OFFSET)
        * 220, (tvecs[i][1] + Y_OFFSET) * 220);
      cv::circle(boatPosImg, tugPt, 2,
        colour[markerIds[i]],
        3, 8, 0);

      if(waypoints[markerIds[i]].x > 0.001 &&
        inWaypMode[markerIds[i]])
      { //Draw line from boat to waypoint in output
        //image
        cv::Point2d waypPt = cv::Point(
          waypoints[markerIds[i]].x
          * 220, waypoints[markerIds[i]].y * 220);
        cv::arrowedLine(videoImg, tugPt, waypPt,
          colour[markerIds[i]], 2, 8, 0, 0.03);
        //Draw circle at waypoint in documentation
        //image
        if(!(counter % 10) )
        { //Draw waypoint every 10 sec
          cv::circle(boatPosImg, waypPt, 20,
            colour[markerIds[i]],
            2, 8, 0);
        }
      }
    }
  }
}

```

---

---

```

    }
}
//Transform rvec to 2D orientation
cv::Rodrigues(rvecs[i], rotation3x3);
double orientation =
rotationMatrixToEulerAngles2D(&rotation3x3);
boat.ID = (uint8_t)markerIds[i];
//x and y assumes corner is at tugboat center
boat.x = tvecs[i][0] + X_OFFSET;
boat.y = tvecs[i][1] + Y_OFFSET;
boat.o = orientation;
if(boat.ID == SHIP_ID)
{
    ship_pub.publish(boat);
}
else
{
    tug_pub.publish(boat);
}
}
}
}
drawShip();
cv::namedWindow("pos", CV_WINDOW_NORMAL);
cv::imshow("pos", boatPosImg);
cv::namedWindow("out", CV_WINDOW_NORMAL);
cv::imshow("out", videoImg);
int key = cv::waitKey(1);

switch(key)
{
    case 27: //Esc aborts the program and captures the
//recording
    {
        saveImage(&boatPosImg);
        return 0;
    }
    case 32: //Spacebar to record recording
    {
        saveImage(&boatPosImg);
        break;
    }
    case 99: //c for "capture", to capture live feed
    {
        saveImage(&videoImg);

```

---

---

```
        break;
    }
    case 114: //r for "reset", reset the recording
    {
        while (!capWebcam.read(boatPosImg)) {}
        break;
    }
}
ros::spinOnce();
loop_rate.sleep();
}
return 0;
}
```

---

```

//TugManager.cpp

#include "ros/ros.h"
#include "tugboat_control/BoatPose.h"
#include "tugboat_control/Waypoint.h"
#include "tugboat_control/PushingForce.h"
#include "tugboat_control/ClearWaypoint.h"
#include "std_msgs/Bool.h"
#include "std_msgs/UInt8MultiArray.h"
#include "std_msgs/UInt8.h"
#include "tugboat_control/addOneTug.h"
#include "tugboat_control/removeOneTug.h"

#include "/usr/local/include/PID_cpp/pid.h"
#include <iostream>
#include <sstream>
#include <unistd.h>

#define SHIP_ID 0
#define STRESS_MODE false //Dynamic tugboat allocation
//using stress levels was only partially implemented
#define MUCH_DISTRESS 100

int distress = 100;
uint8_t orderIDcount = 0;

std_msgs::UInt8MultiArray waypTugs;
std_msgs::UInt8MultiArray ctrlTugs;
std::vector<tugboat_control::BoatPose> tugRequests;
//In ship coordinates (x is along ship), ID = order ID,
// not tug ID
tugboat_control::BoatPose shipPose;
tugboat_control::Waypoint shipWaypoint;

void distressCallback(const std_msgs::Bool::ConstPtr&
    stressed)
{
    if(stressed->data) {
        distress++;
    } else {
        distress--;
    }
}

void respondToDistress(ros::ServiceClient *remove_client,

```

---

```

tugboat_control::removeOneTug *remove_srv,
ros::ServiceClient *add_client,
tugboat_control::addOneTug *add_srv)
{
    if (distress > MUCH_DISTRESS * (tugRequests.size()
    + 1) && waypTugs.data.size() > tugRequests.size()
    ) {
        std::cout << "Requesting new tugboat to Ship
        Control\n";
        if(add_client->call(*add_srv)){
            //Add to tugRequests
            tugboat_control::BoatPose tempRequest =
            add_srv->response.Pose;
            tempRequest.ID = ++orderIDcount;
            tugRequests.push_back(tempRequest);
            std::cout << "Request approved. " <<
            tugRequests.size() << " tugs on their way to
            support ship\n";
        }
        else
        {
            std::cout << "Request denied\n";
        }
    } else if(distress < MUCH_DISTRESS *
    (tugRequests.size() - 1)) {
        if(remove_client->call(*remove_srv)){
            std::cout << "Removing tugboat " <<
            (int)remove_srv->response.ID << " from Ship
            Control\n";
            //remove Tugboat from ctrlTugs, add to waypTugs
            for (uint8_t i = 0; i < ctrlTugs.data.size(); ++i)
            {
                if(ctrlTugs.data[i] == remove_srv->response.ID){
                    ctrlTugs.data.erase(ctrlTugs.data.begin() + i);
                    waypTugs.data.push_back(remove_srv->response.ID);
                    break;
                }
            }
        }
    }
}

void waypClearCallback(const tugboat_control::
    ClearWaypoint::ConstPtr& msg)
{ //Clear tugRequest, give responsibillity from

```

---



---

```

// Waypoint to ShipControl
std::cout << "Tugboat " << (int)msg->tugID <<
" has reached its final waypoint. Assigning to
Ship Control.\n";
for (int i = 0; i < tugRequests.size(); ++i)
{
    if(tugRequests[i].ID == msg->orderID){
        tugRequests.erase(tugRequests.begin() + i);
        break;
    }
}
for (int i = 0; i < waypTugs.data.size(); ++i)
{
    if(waypTugs.data[i] == msg->tugID){
        waypTugs.data.erase(waypTugs.data.begin() + i);
        ctrlTugs.data.push_back(msg->tugID);
        break;
    }
}
}

void ctrlClearCallback(const std_msgs::UInt8::ConstPtr&
tugID)
{ //Clear from ShipControl for tugs that loose contact
// with shi etc
for (int tug = 0; tug < ctrlTugs.data.size(); ++tug)
{
    if(ctrlTugs.data[tug] == tugID->data)
    {
        std::cout << "Tugboat " << (int)tugID->data <<
" has strayed too far from the ship. Assigning
to Waypoint Control.\n";
        ctrlTugs.data.erase(ctrlTugs.data.begin() + tug);
        waypTugs.data.push_back(tugID->data);
    }
}
}

tugboat_control::Waypoint shipToWorldCoordinates(
tugboat_control::BoatPose poseIn)
{
    tugboat_control::Waypoint wayp;
    wayp.ID = poseIn.ID;
    wayp.v = 0;
    wayp.o = 0;

```

---

---

```

double sin0 = sin(shipPose.o);
double cos0 = cos(shipPose.o);

wayp.x = (cos0 * poseIn.x - sin0 * poseIn.y) +
shipPose.x;
wayp.y = (sin0 * poseIn.x + cos0 * poseIn.y) +
shipPose.y;
return wayp;
}

void shipPoseCallback(const tugboat_control::BoatPose::
  ConstPtr& pose_in)
{
  shipPose = *pose_in;
}

void sendTugRequests(ros::Publisher *pub)
{ //Take BoatPose tugRequests, transform to world
  //coordinate system, and send as waypoint to Waypoint manager
  tugboat_control::Waypoint wayp;
  for (int tug = 0; tug < tugRequests.size(); ++tug)
  {
    wayp = shipToWorldCoordinates(tugRequests[tug]);
    pub->publish(wayp);
    std::cout << "Sent waypoint request to: x=" <<
    wayp.x << "\t y=" << wayp.y << "\n";
  }
  std::cout << "\n";
}

void startupCallback(const std_msgs::UInt8::ConstPtr&
  IDIn)
{
  waypTugs.data.push_back(IDIn->data);
  std::cout << "Starting new tugboat with id " <<
  (int)IDIn->data << "\n";
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "manager");
  ros::NodeHandle n;

  ros::ServiceClient remove_client = n.serviceClient<

```

---

---

```

tugboat_control::removeOneTug>("removeOneTug");
tugboat_control::removeOneTug remove_srv;

ros::ServiceClient add_client = n.serviceClient<
tugboat_control::addOneTug>("addOneTug");
tugboat_control::addOneTug add_srv;

ros::Subscriber pose_sub = n.subscribe(
"shipPose", 100, shipPoseCallback);
//Get position of ship
ros::Subscriber distress_sub = n.subscribe(
"distress", 1000, distressCallback); //
ros::Subscriber waypClear_sub = n.subscribe(
"clearWaypoint", 100, waypClearCallback); //
ros::Subscriber startup_sub = n.subscribe(
"startup", 100, startupCallback); //
ros::Subscriber ctrlClear_sub = n.subscribe(
"ctrlClear", 100, ctrlClearCallback); //

ros::Publisher waypTugs_pub = n.advertise<
std_msgs::UInt8MultiArray>("waypTugs", 1);
//List of tugboats controlled by Waypoints
ros::Publisher ctrlTugs_pub = n.advertise<
std_msgs::UInt8MultiArray>("ctrlTugs", 1);
//List of tugboats controlled by Ship Control
ros::Publisher waypReq_pub = n.advertise<
tugboat_control::Waypoint>("waypointRequest", 1000);
//Requests tugboats at specific locations from Waypoint
// manager
ros::Publisher shipWayp_pub = n.advertise<
tugboat_control::Waypoint>("shipWaypoint", 1);

shipWaypoint.ID = SHIP_ID;
shipWaypoint.x = 1;
shipWaypoint.y = 1;
shipWaypoint.o = 0;
shipWaypoint.v = 0.05;

ros::Rate loop_rate(1);

std::cout << "Tug Manager node initialized
successfully\n";

while (ros::ok())
{

```

---

---

```

ros::spinOnce();
waypTugs_pub.publish(waypTugs);
ctrlTugs_pub.publish(ctrlTugs);
shipWayp_pub.publish(shipWaypoint); //for
illustration purposes only

if(STRESS_MODE)
{
    std::cout << "Stress level: " << distress << "\n";
    respondToDistress(&remove_client, &remove_srv,
        &add_client, &add_srv);
}
else if(waypTugs.data.size() > tugRequests.size())
{ //All tugboats added to control
    std::cout << "Adding one tugboat to ship control... ";
    if(add_client.call(add_srv)){
        //Add to tugRequests
        tugboat_control::BoatPose tempRequest =
            add_srv.response.Pose;
        tempRequest.ID = ++orderIDcount;
        tugRequests.push_back(tempRequest);
        std::cout << "Request approved. " <<
            tugRequests.size() << " tugs shipbound\n";
    }
    else
    {
        std::cout << "Request denied\n";
    }
}
sendTugRequests (&waypReq_pub);
std::cout << "numCtrlTugs: " << ctrlTugs.data.size()
<< "\tnumWaypTugs: " << waypTugs.data.size() << "\n";

loop_rate.sleep();
}
return 0;
}

```

---

```

/* WaypointManager.cpp
   This module is a placeholder for Cox' path-planning
   algorithms, but is included to show the interface and
   the general functionality. Not that this module
   has no path-planning capabilities.*/
#include "ros/ros.h"
#include "tugboat_control/BoatPose.h"
#include "tugboat_control/Waypoint.h"
#include "tugboat_control/ClearWaypoint.h"
#include "std_msgs/UInt8MultiArray.h"

#include <iostream>
#include <sstream>
#include <unistd.h>
#include <vector>

#define SHIP_ID 0
#define ACCEPTABLE_DIST_ERROR 0.3 //30 cm is ok for now

tugboat_control::BoatPose shipPose;
tugboat_control::Waypoint shipWaypoint;

//tugPoses and waypTugIDs have the same indexes for
//the same tugboats
std_msgs::UInt8MultiArray waypTugIDs;
std::vector<tugboat_control::BoatPose> tugPoses;
std::vector<tugboat_control::Waypoint> tugWaypoints;
//ID = orderID

void shipPoseCallback(const tugboat_control::BoatPose::
  ConstPtr& pose_in)
{ //Not used in dummy
  shipPose = *pose_in;
}

void tugPoseCallback(const tugboat_control::BoatPose::
  ConstPtr& pose_in)
{
  for (int tug = 0; tug < waypTugIDs.data.size(); ++tug)
  {
    if(pose_in->ID == waypTugIDs.data[tug]){
      tugPoses[tug] = *pose_in;
      return;
    }
  }
}

```

---

---

```

}

void waypTugsCallback(const std_msgs::UInt8MultiArray::
  ConstPtr& tugsIn)
{
  std::vector<tugboat_control::BoatPose> newPoses;

  waypTugIDs = *tugsIn;
  for (int ID = 0; ID < waypTugIDs.data.size(); ++ID)
  {
    for (int pose = 0; pose < tugPoses.size(); ++pose)
    {
      if(tugPoses[pose].ID == waypTugIDs.data[ID])
      {
        newPoses.push_back(tugPoses[pose]);
        break;
      }
    }
    if(newPoses.size() <= ID)
    { //No element added above, because no Pose is in
      //record
      tugboat_control::BoatPose blankPose;
      blankPose.ID = waypTugIDs.data[ID];
      blankPose.x = -10;
      blankPose.y = 0;
      blankPose.o = 0;
      newPoses.push_back(blankPose);
    }
  }
  tugPoses = newPoses;
}

void waypReqCallback(const tugboat_control::Waypoint::
  ConstPtr& waypIn)
{
  for (int wayp = 0; wayp < tugWaypoints.size(); ++wayp)
  {
    if(tugWaypoints[wayp].ID == waypIn->ID){
      tugWaypoints[wayp] = *waypIn;
      return;
    }
  }
  tugWaypoints.push_back(*waypIn);
}

```

---

---

```

int main(int argc, char **argv)
{
    ros::init(argc, argv, "dummyWaypoint");

    ros::NodeHandle n;
    ros::Publisher clearWayp_pub = n.advertise<
tugboat_control::ClearWaypoint>("clearWaypoint", 1000);
    ros::Publisher tugWayp_pub = n.advertise<
tugboat_control::Waypoint>("waypoint", 10);
    ros::Publisher shipWayp_pub = n.advertise<
tugboat_control::Waypoint>("shipWaypoint", 10);
    ros::Subscriber shipPose_sub = n.subscribe(
    "shipPose", 100, shipPoseCallback);
    ros::Subscriber tugPose_sub = n.subscribe(
    "pose", 100, tugPoseCallback);
    ros::Subscriber waypTugs_sub = n.subscribe(
    "waypTugs", 100, waypTugsCallback);
    ros::Subscriber waypReq_sub = n.subscribe(
    "waypointRequest", 100, waypReqCallback);

    ros::Rate loop_rate(1);

    shipWaypoint.ID = SHIP_ID;
    shipWaypoint.x = 1;
    shipWaypoint.y = 1;
    shipWaypoint.o = 0;
    shipWaypoint.v = 0.05;

    std::cout << "Waypoint Dummy node initialized
successfully\n";
    while (ros::ok())
    {
        ros::spinOnce();
        for (int waypoint = 0; waypoint <
tugWaypoints.size(); ++waypoint)
        {
            for (int tug = 0; tug < waypTugIDs.data.size();
            ++tug)
            {
                if(sqrt( pow(tugWaypoints[waypoint].x -
tugPoses[tug].x, 2) + pow(tugWaypoints[
waypoint].y - tugPoses[tug].y, 2) ) <
ACCEPTABLE_DIST_ERROR)
                {
                    tugboat_control::ClearWaypoint msg;

```

---

```

msg.tugID = waypTugIDs.data[tug];
msg.orderID = tugWaypoints[waypoint].ID;
clearWayp_pub.publish(msg);

std::cout << "Clearing waypoint with id " <<
(int)msg.orderID << "\n";

tugWaypoints.erase(tugWaypoints.begin() +
    waypoint);
waypTugIDs.data.erase(waypTugIDs.data.begin()
    + tug);
tugPoses.erase(tugPoses.begin() + tug);
break;
    }
}
}
//Publish all remaining waypoints
shipWayp_pub.publish(shipWaypoint);

tugboat_control::Waypoint tempWayp;
for (int tug = 0; tug < waypTugIDs.data.size();
    ++tug)
{
    if (tug < tugWaypoints.size())
    {
        tempWayp = tugWaypoints[tug];
        std::cout << "Tugboat " <<
            (int)waypTugIDs.data[tug] << "to ( " <<
            tugWaypoints[tug].x << ", " <<
            tugWaypoints[tug].y << " )\n";
    }
    else
    { //No waypoint for you -> stay put
        tempWayp.x = tugPoses[tug].x;
        tempWayp.y = tugPoses[tug].y;
        tempWayp.o = tugPoses[tug].o;
        tempWayp.v = 0;
        std::cout << "Tugboat " <<
            (int)waypTugIDs.data[tug] << " stay put\n";
    }
    tempWayp.ID = waypTugIDs.data[tug];
    tugWayp_pub.publish(tempWayp);
}
std::cout << "\n";
loop_rate.sleep();

```

---



---

```
    }  
  
    return 0;  
}
```

---

```

//ShipControl.cpp

#include "ros/ros.h"
#include "tugboat_control/BoatPose.h"
#include "tugboat_control/Waypoint.h"
#include "tugboat_control/PushingForce.h"
#include "tugboat_control/TugSetpoints.h"
#include "std_msgs/Bool.h"
#include "std_msgs/UInt8MultiArray.h"
#include "std_msgs/UInt8.h"
#include "tugboat_control/addOneTug.h"
#include "tugboat_control/removeOneTug.h"

#include "/usr/local/include/PID_cpp/pid.h"
#include <iostream>
#include <sstream>
#include <unistd.h>
#include <vector>
#include <algorithm>
#include <cmath>

#define SHIP_ID 0
#define HALF_SHIP_WIDTH 0.35
//plus one tugboat radius + margin
#define HALF_SHIP_LENGTH 0.40
//plus one tugboat radius + margin
#define ACCEPTABLE_DIST_ERROR 1.0
//0.5 m from ship means tug has lost contact
#define MAX_PUSHING_FORCE 0.5
//Max setpoint for pushing force, in N
#define MIN_PUSHING_FORCE 0.1
//Min force required to maintain contact, in N

uint8_t orderID = 0;
std_msgs::UInt8MultiArray ctrlTugIDs;
std::vector<tugboat_control::BoatPose> tugPoses;
//In ship coordinates (x is along ship)
tugboat_control::BoatPose shipPose;
tugboat_control::Waypoint shipWaypoint;
std_msgs::UInt8MultiArray discardTugs;

void shipPoseCallback(const tugboat_control::BoatPose::
    ConstPtr& pose_in)
{
    shipPose = *pose_in;

```

---

```

}

tugboat_control::BoatPose worldToShipCoordinates(
    tugboat_control::BoatPose poseIn)
{ //Transforms from world coordinates to ship coordinates
    tugboat_control::BoatPose poseOut;
    poseOut.ID = poseIn.ID;
    poseOut.o = PIDnormalizeAngle( poseIn.o - shipPose.o );

    double dx = poseIn.x - shipPose.x;
    double dy = poseIn.y - shipPose.y;
    double sin0 = sin(shipPose.o);
    double cos0 = cos(shipPose.o);

    //Rotation matrix opposite of normal because the
    // coordinate system is left-handed, thus rotation
    // is clockwise
    poseOut.x = cos0 * dx + sin0 * dy;
    poseOut.y = - sin0 * dx + cos0 * dy;
    return poseOut;
}

void tugPoseCallback(const tugboat_control::
    BoatPose::ConstPtr& pose_in)
{
    for (int tug = 0; tug < ctrlTugIDs.data.size(); ++tug)
    {
        if(pose_in->ID == ctrlTugIDs.data[tug]){
            tugPoses[tug] = worldToShipCoordinates(*pose_in);
            if(sqrt( pow(tugPoses[tug].x, 2) +
                pow(tugPoses[tug].y, 2) ) >
                ACCEPTABLE_DIST_ERROR)
            {
                //Tugboat lost touch with ship -
                //send back to waypoint manager
                discardTugs.data.push_back(ctrlTugIDs.data[tug]);
                std::cout << "Tugboat " <<
                    (int)ctrlTugIDs.data[tug] <<
                    " strayed too far from ship. Assumed lost.\n";
            }
            return;
        }
    }
}

void waypCallback(const tugboat_control::Waypoint::

```

---

---

```

    ConstPtr& wayp_in)
{
    shipWaypoint = *wayp_in;
}

void ctrlTugsCallback(const std_msgs::UInt8MultiArray::
    ConstPtr& tugsIn)
{
    std::vector<tugboat_control::BoatPose> newPoses;

    ctrlTugIDs = *tugsIn;
    for (int ID = 0; ID < ctrlTugIDs.data.size(); ++ID)
    {
        for (int pose = 0; pose < tugPoses.size(); ++pose)
        {
            if(tugPoses[pose].ID == ctrlTugIDs.data[ID])
            {
                newPoses.push_back(tugPoses[pose]);
                break;
            }
        }
        if(newPoses.size() <= ID)
        { //No element added above, because no Pose
          // is in record
            tugboat_control::BoatPose blankPose;
            blankPose.ID = ctrlTugIDs.data[ID];
            blankPose.x = 0;
            blankPose.y = -10;
            blankPose.o = 0;
            newPoses.push_back(blankPose);
            std::cout << "Adding tugboat " <<
                (int)ctrlTugIDs.data[ID] << " to Ship Control. +n"; // << --numTug
        }
    }
    tugPoses = newPoses;
}

//Helper variables and functions
//START_UNIQUE
int startUpPositioningVariable = 0;

int determineShipSide(tugboat_control::BoatPose tugPose)
    //tugPose in ship coordinates
{ //Sides are: 0 for front, 3 for port, 2 for aft,
  // 1 for starboard

```

---

---

```

double tugPosAngle = atan2(tugPose.y, tugPose.x);
//angle from origin to tug position in ship coordinates
double cornerAngle = atan2(HALF_SHIP_WIDTH,
    HALF_SHIP_LENGTH);
std::cout << "tugPosAngle: " << tugPosAngle <<
" cornerAngle: " << cornerAngle << "\n";
if(std::abs(tugPosAngle) < cornerAngle)
{
    return 0;
}
else if(std::abs(tugPosAngle) > (3.14 - cornerAngle))
{
    return 2;
}
else if(tugPosAngle > 0)
{
    return 1;
}
else
{
    return 3;
}
}
//END_UNIQUE

bool addOneTug(tugboat_control::addOneTug::Request
    &req, tugboat_control::addOneTug::Response &res)
{
    //If one extra tugboat is needed - where does it go?
    //Pose given in ship coordinates
    std::cout << "Adding one tugboat...\t";

    //START_UNIQUE
    //Tugboats are arrayed around ship, as evenly as
    //possible
    int shipSide;
    if(startUpPositioningVariable < 4)
    { //First 4 tugboats should go to opposing sides
        shipSide = startUpPositioningVariable++;
    }
    else
    {
        std::vector<int> occupancy(4);
        double tugPosAngle;
        //Count number of tugboats at each ship side
        for (int tug = 0; tug < ctrlTugIDs.data.size(); ++tug)

```

---

```

    {
        occupancy[determineShipSide(tugPoses[tug])]++;
    }
    //Send tugboat to ship side with the fewest tugboats
    std::vector<int>::iterator it = std::min_element(
        occupancy.begin(), occupancy.end());
    shipSide = std::distance(occupancy.begin(), it);
}

switch(shipSide)
{
    case 0:
    {
        res.Pose.x = HALF_SHIP_LENGTH;
        res.Pose.y = 0;
        break;
    }
    case 1:
    {
        res.Pose.x = 0;
        res.Pose.y = HALF_SHIP_WIDTH;
        break;
    }
    case 2:
    {
        res.Pose.x = -HALF_SHIP_LENGTH;
        res.Pose.y = 0;
        break;
    }
    case 3:
    {
        res.Pose.x = 0;
        res.Pose.y = -HALF_SHIP_WIDTH;
        break;
    }
}

std::cout << "Requesting tugboat on side " <<
shipSide << " (0: stern, 1: starboard, 2: aft, 3:
port).\n";
res.Pose.ID = orderID++;
    res.Pose.o = 0; //Not used
    //END_UNIQUE
std::cout << "Requested tugboat at: x=" <<
res.Pose.x << "\t y=" << res.Pose.y << "\n";
    return true;

```

---

---

```

}

bool removeOneTug (tugboat_control::removeOneTug::Request
&req, tugboat_control::removeOneTug::Response &res)
{
    //If there are too many tugboats, one is removed
    if(ctrlTugIDs.data.size() > 0){
        //START_UNIQUE
        res.ID = ctrlTugIDs.data[ctrlTugIDs.data.size()];
        //This algorithm simply selects the last tugboat
        //on the list
        //END_UNIQUE
        std::cout << "Removing tugboat " << (int)res.ID <<
" from Ship Control\n";
        return true;
    }
    else
    {
        return false;
    }
}

void control(ros::Publisher *pub)
{
    //This is where the magic happens
    //Find out where the ship should be going
    double shipSetHeading = atan2(shipWaypoint.y
- shipPose.y, shipWaypoint.x - shipPose.x );
    //START_UNIQUE
    tugboat_control::TugSetpoints ctrl;
    for (int tug = 0; tug < ctrlTugIDs.data.size(); ++tug)
    {
        ctrl.ID = ctrlTugIDs.data[tug];
        std::cout << "Tug " << (int)ctrl.ID <<
" at shipSide " << determineShipSide(tugPoses[tug])
<< " in ship coordinates ( " << tugPoses[tug].x
<< ", " << tugPoses[tug].y << " )\n";
        double oSetShipCoordinates = -3.14 + 1.57 *
determineShipSide(tugPoses[tug]);

        ctrl.o = PIDnormalizeAngle(oSetShipCoordinates +
shipPose.o);
        ctrl.force = std::max(MIN_PUSHING_FORCE,
MAX_PUSHING_FORCE * cos(ctrl.o - shipSetHeading) );
        pub->publish(ctrl);
    } //END_UNIQUE
}

```

---

---

```

int main(int argc, char **argv)
{
    ros::init(argc, argv, "generic_control");
    ros::NodeHandle n;
    ros::ServiceServer remove_service =
    n.advertiseService("removeOneTug", removeOneTug);
    ros::ServiceServer add_service =
    n.advertiseService("addOneTug", addOneTug);
    ros::Subscriber wayp_sub =
    n.subscribe("shipWaypoint", 1, waypCallback);
    ros::Subscriber shipPose_sub =
    n.subscribe("shipPose", 100, shipPoseCallback);
    ros::Subscriber tugPose_sub =
    n.subscribe("pose", 100, tugPoseCallback);
    ros::Subscriber ctrlTugs_sub =
    n.subscribe("ctrlTugs", 100, ctrlTugsCallback);
    ros::Publisher ctrl_pub =
    n.advertise<tugboat_control::TugSetpoints>(
        "control", 100);
    ros::Publisher stress_pub =
    n.advertise<std_msgs::Bool>("distress", 100);
    ros::Publisher discard_pub =
    n.advertise<std_msgs::UInt8>("ctrlClear", 100);

    ros::Rate loop_rate(10);

    shipWaypoint.ID = SHIP_ID;
    shipWaypoint.x = 1;
    shipWaypoint.y = 1;
    shipWaypoint.o = 0;
    shipWaypoint.v = 0.05;

    std::cout << "Ship Control node initialized
    successfully\n";

    while (ros::ok())
    {
        ros::spinOnce();
        if(ctrlTugIDs.data.size() > 0)
        {
            control(&ctrl_pub);
        }
        else {
            std_msgs::Bool msg;

```



---

```
        msg.data = true;
        stress_pub.publish(msg);
    }

    while( discardTugs.data.size() > 0)
    {
std::cout << "Discarding tugboat " <<
(int)discardTugs.data[0] << " from Ship Control\n";

std_msgs::UInt8 msg;
        msg.data = discardTugs.data[0];
        discard_pub.publish(msg);
        discardTugs.data.erase(discardTugs.data.begin());
std::cout << "Debug: Discard OK\n";
    }
    loop_rate.sleep();
}
return 0;
}
```

---

```
//TugController.cpp

#include "ros/ros.h"
#include "tugboat_control/Thrust.h"
#include "tugboat_control/TugSetpoints.h"
#include "tugboat_control/Waypoint.h"
#include "tugboat_control/BoatPose.h"
#include "tugboat_control/PushingForce.h"
#include "std_msgs/Bool.h"
#include "std_msgs/UInt8.h"

#include "/usr/local/include/PID_cpp/pid.h"
#include <string>
#include <iostream>
#include <sstream>
#include <unistd.h>
#include <cmath>

#define NO_CONTROL 0
#define CTRL 1
#define WAYP 2
#define ACCEPTABLE_ANGLE_ERROR 0.5
// 0.5 rad = 29 degrees
#define ACCEPTABLE_DIST_ERROR 0.0
// 0 cm from waypoint is considered OK
#define TIMEOUT_TIME 1000 //ms

#define dt 0.1
//Controller timestep in seconds
#define output_max 100
//Output is measured in %
#define output_min -100
#define circular true
#define noncircular false

#define ctrl_ccwturn_Kp 15
#define ctrl_ccwturn_Kd 30
#define ctrl_ccwturn_Ki 0

#define PUSHING_CONTROLLER_ENABLED false
#define FORCE_TO_THRUST 100.0
#define ctrl_thrust_Kp 10
#define ctrl_thrust_Kd 10
#define ctrl_thrust_Ki 0
```

---

```

#define wayp_ccwturn_Kp      15
#define wayp_ccwturn_Kd      30
#define wayp_ccwturn_Ki      0

//Temporary disabled
#define SPEED_CONTROLLER_ENABLED false
#define wayp_thrust_Kp      1000
#define wayp_thrust_Kd      100
#define wayp_thrust_Ki      0

PID ctrl_ccwturnPID = PID(dt, output_max,
    output_min, ctrl_ccwturn_Kp, ctrl_ccwturn_Kd,
    ctrl_ccwturn_Ki, circular);
PID ctrl_thrustPID = PID(dt, output_max, 0,
    ctrl_thrust_Kp, ctrl_thrust_Kd, ctrl_thrust_Ki,
    noncircular);

PID wayp_ccwturnPID = PID(dt, output_max, output_min,
    wayp_ccwturn_Kp, wayp_ccwturn_Kd, wayp_ccwturn_Ki,
    circular);
PID wayp_thrustPID = PID(dt, 50, 0, wayp_thrust_Kp,
    wayp_thrust_Kd, wayp_thrust_Ki, noncircular);

tugboat_control::BoatPose pose;
tugboat_control::BoatPose lastPose;
tugboat_control::Waypoint wayp;
tugboat_control::Thrust cmd;
tugboat_control::PushingForce push;
tugboat_control::TugSetpoints ctrl;

uint8_t id; //Passed as an argument. Default value 0
int controlMode = NO_CONTROL;
bool timeout = false;
long lastPoseTime = 0;
double v;

void resetController()
{
    if(controlMode == CTRL)
    {
        ctrl_ccwturnPID.resetPID();
        ctrl_thrustPID.resetPID();
    }
    else if (controlMode == WAYP)
    {

```

---

```

        wayp_ccwturnPID.resetPID();
        wayp_thrustPID.resetPID();
    }
}

void poseCallback(const tugboat_control::BoatPose::
    ConstPtr& pose_in)
{
    if(pose_in->ID == id){
        pose = *pose_in;
        long thisPoseTime = PIDmsNow();

        if(controlMode == WAYP && SPEED_CONTROLLER_ENABLED){
            //need to update speed also
            double dx = pose.x - lastPose.x;
            double dy = pose.y - lastPose.y;
            double directionOfTravel = atan2(dy, dx);
            double angleDifference =
                PIDnormalizeAngle(pose.o - directionOfTravel);

            double timechange = thisPoseTime - lastPoseTime;
            v = sqrt(dx*dx + dy*dy) * cos(angleDifference)
                / (timechange/1000);
        }

        lastPoseTime = thisPoseTime;
        lastPose = pose;
    }
}

void pushCallback(const tugboat_control::PushingForce::
    ConstPtr& push_in)
{
    if(push_in->ID == id){
        push = *push_in;
    }
}

void waypCallback(const tugboat_control::Waypoint::
    ConstPtr& wayp_in)
{
    if(wayp_in->ID == id){
        if(controlMode != WAYP){
            std::cout << "Enter Waypoint Mode\n";
            resetController();
        }
    }
}

```

---

---

```

        controlMode = WAYP;
    }
    wayp = *wayp_in;
}
}

void ctrlCallback(const tugboat_control::TugSetpoints::
ConstPtr& ctrl_in)
{
    if(ctrl_in->ID == id){
        if(controlMode != CTRL){
            std::cout << "Enter Control Mode\n";
            resetController();
            controlMode = CTRL;
        }
        ctrl = *ctrl_in;
    }
}

void computeControl(bool mode, ros::Publisher stress_pub)
{
    long timeSinceLastPoseMeasurement = PIDmsNow()
- lastPoseTime;
    if( timeSinceLastPoseMeasurement < TIMEOUT_TIME)
    {
        if(controlMode == WAYP)
        {
            double dist = sqrt(pow(pose.x - wayp.x, 2)
+ pow(pose.y - wayp.y, 2));
            double oSet = atan2(wayp.y - pose.y, wayp.x
- pose.x);
            double oErr = PIDnormalizeAngle(oSet - pose.o);

            if(std::abs(oErr) < ACCEPTABLE_ANGLE_ERROR
&& dist > ACCEPTABLE_DIST_ERROR)
            {
                if(SPEED_CONTROLLER_ENABLED){
                    cmd.thrust = (int8_t)wayp_thrustPID.calculate(
wayp.v, v );
                }
                else if(dist < 1){
                    cmd.thrust = (int8_t)(dist * 40 + 10);
                }
                else {
                    cmd.thrust = 50;
                }
            }
        }
    }
}

```

---

---

```

    }
  }
  else
  {
    cmd.thrust = 0;
  }
  cmd.ccwturn = (int8_t)wayp_ccwturnPID.calculate(
    oSet, pose.o);
  return;
}

else if(controlMode == CTRL)
{
  cmd.ccwturn = (int8_t)ctrl_ccwturnPID.calculate(
    ctrl.o, pose.o);

  double oErrCtrl = PIDnormalizeAngle(
    ctrl.o - pose.o);
  if( std::abs( oErrCtrl ) < ACCEPTABLE_ANGLE_ERROR )
  {
    if(PUSHING_CONTROLLER_ENABLED)
    { //Feedforward + feedback
      cmd.thrust = (int8_t)( ctrl.force
        * FORCE_TO_THRUST + ctrl_thrustPID.calculate(
          ctrl.force, push.force) );
    }
    else
    { //Only feedforward
      cmd.thrust = (int8_t)(FORCE_TO_THRUST *
        ctrl.force);
    }
  }
  //For stress-based tugboat managing
  std_msgs::Bool stress;
  if(cmd.thrust > 60)
  {
    stress.data = true;
    stress_pub.publish(stress);
  }
  else if(cmd.thrust < 20)
  {
    stress.data = false;
    stress_pub.publish(stress);
  }
  return;
}

```

---

---

```

    }
}
//More than TIMEOUT_TIME ms since last pos update,
//or no control mode - shut down tugboat
cmd.thrust = 0;
cmd.ccwturn = 0;
resetController();
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "tug");
    ros::NodeHandle n;
    ros::Subscriber pose_sub = n.subscribe(
        "pose", 100, poseCallback);
    ros::Subscriber ctrl_sub = n.subscribe(
        "control", 100, ctrlCallback);
    ros::Subscriber wayp_sub = n.subscribe(
        "waypoint", 100, waypCallback);
    ros::Subscriber push_sub = n.subscribe(
        "pushingForce", 100, pushCallback);

    ros::Publisher cmd_pub = n.advertise<tugboat_control::
    Thrust>("thrust", 100);
    ros::Publisher stress_pub = n.advertise<std_msgs::
    Bool>("distress", 100);
    ros::Publisher startup_pub = n.advertise<std_msgs::
    UInt8>("startup", 100);

    ros::Rate loop_rate(10);

    ros::NodeHandle pnh("~");
    double ID;
    if(pnh.getParam("ID", ID)) {
        id = (uint8_t)ID;
    }
    else {
        id = 1;
    }

    cmd.ID = id;
    std_msgs::UInt8 idMsg;
    idMsg.data = id;
    if(ros::ok())
    {

```

---

```
    ros::Duration(0.5).sleep();
    startup_pub.publish(idMsg);
    ros::spinOnce();
}

std::cout << "TugController initialized
successfully with id " << (int)id << "\n";

while (ros::ok())
{
    ros::spinOnce();
    computeControl(controlMode, stress_pub);
    cmd_pub.publish(cmd);
    loop_rate.sleep();
}

return 0;
}
```



---

```

//ROS_RadioTx.ino

#include <ros.h>
#include <tugboat_control/Thrust.h>
#include <std_msgs/Int16.h>

#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
const uint64_t txPipe = 0xF0F0F0F0E1LL;

#define OK_LED A2
#define NOT_OK_LED A3

void transmitThrust(const tugboat_control::Thrust& cmd)
{
    radio.stopListening();
    bool ok = radio.write( &cmd, sizeof(cmd) );
    if(ok)
    {
        OK(true);
    }
    else
    {
        OK(false);
    }
    radio.startListening();
}

ros::NodeHandle nh;
tugboat_control::Thrust cmd;
ros::Subscriber<tugboat_control::Thrust> sub_thrust(
    "thrust", &transmitThrust);
std_msgs::Int16 percentloss;
float success = 0;
float fail = 0;
ros::Publisher pub_com("messageLoss", &percentloss);

void OK(bool isOK){
    if(isOK){
        digitalWrite(OK_LED, HIGH);
        digitalWrite(NOT_OK_LED, LOW);
    }
}

```

---

```
    success += 1.0;
}
else {
    digitalWrite(OK_LED, LOW);
    digitalWrite(NOT_OK_LED, HIGH);
    fail += 1.0;
}
}

void setup()
{
    //Serial.begin(57600);

    nh.initNode();
    nh.subscribe(sub_thrust);
    nh.advertise(pub_com);

    pinMode(OK_LED, OUTPUT);
    pinMode(NOT_OK_LED, OUTPUT);
    OK(false);

    radio.begin();
    radio.setRetries(15,15);
    radio.openWritingPipe(txPipe);
    radio.startListening();
    percentloss.data = 0;
    OK(true);

}

void loop()
{
    percentloss.data = (int long) (1000.0*fail / (fail
    + success));
    pub_com.publish(&percentloss);
    nh.spinOnce();
    delay(10);
}
```

---

```

//ROS_RadioRx.ino

#include <ros.h>
#include <tugboat_control/PushingForce.h>

#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

ros::NodeHandle nh;
tugboat_control::PushingForce push;
ros::Publisher pub_push("pushingForce", &push);

RF24 radio(8,7);
const uint64_t rxPipe = 0xF0F0F0F0D2LL;

#define OK_LED A2
#define NOT_OK_LED A3
#define WAIT_TIME 100 //ms

void OK(bool isOK){
    if(isOK){
        digitalWrite(OK_LED, HIGH);
        digitalWrite(NOT_OK_LED, LOW);
    }
    else {
        digitalWrite(OK_LED, LOW);
        digitalWrite(NOT_OK_LED, HIGH);
    }
}

void setup()
{
    nh.initNode();
    nh.advertise(pub_push);

    radio.begin();
    radio.setRetries(15,15);
    radio.openReadingPipe(1,rxPipe);
    radio.startListening();

    pinMode(OK_LED, OUTPUT);
    pinMode(NOT_OK_LED, OUTPUT);
    OK(false);
}

```

---

```
void loop()
{
    unsigned long started_waiting_at = millis();
    bool timeout = false;
    while ( ! radio.available() && ! timeout )
    {
        if (millis() - started_waiting_at > WAIT_TIME)
        {
            timeout = true;
        }
    }

    if( timeout)
    {
        OK(false);
    }
    else
    {
        bool done = false;
        tugboat_control::PushingForce got_push;
        while(!done){
            done = radio.read( &got_push, sizeof(
                tugboat_control::PushingForce) );
        }
        radio.stopListening();
        radio.startListening();
        OK(true);
        push = got_push;
        pub_push.publish( &push );
    }
    nh.spinOnce();
}
```

---

```

//TugOS.ino

#define TUG_ID 3
// Needs to be different for each tugboat

// Might be different for each tugboat
#define leftMotorForward 5
#define leftMotorBackward 6
#define rightMotorForward 9
#define rightMotorBackward 10

#define deadband 30
//smallest motor output allowed, to avoid damaging motor
#define maxout 255-deadband

#include <ros.h>
#include <tugboat_control/PushingForce.h>
#include <tugboat_control/Thrust.h>

tugboat_control::Thrust cmd;
tugboat_control::PushingForce push;

#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
const uint64_t rxPipe = 0xF0F0F0F0E1LL;
const uint64_t txPipe = 0xF0F0F0F0D2LL;

#define OK_LED A2
#define NOT_OK_LED A3
#define WAIT_TIME 100 //ms

#include "HX711.h"
// HX711.DOUT - pin #A1, HX711.PD_SCK - pin #A0
HX711 scale(A1, A0);
// parameter "gain" is ommited; the default value
//128 is used by the library
#define SCALE -200591.f //To get N

int leftMotor = 0;
int rightMotor = 0;
#define dm 20 //max change in motor output per step

```

---

*//Functions*

```
bool getNewCommand()
{
    radio.startListening();
    unsigned long started_waiting_at = millis();
    bool timeout = false;
    while ( ! radio.available() && ! timeout )
    {
        if (millis() - started_waiting_at > WAIT_TIME)
        {
            timeout = true;
        }
    }

    if( timeout)
    {
        //OK(false);
        cmd.thrust = 0;
        cmd.ccwturn = 0;
    }
    else
    { // New radio message

        bool done = false;
        tugboat_control::Thrust newcmd;
        while(!done){
            done = radio.read( &newcmd, sizeof(
                tugboat_control::Thrust) );
        }

        radio.stopListening();

        //OK(true);
        if(newcmd.ID == TUG_ID)
        {
            cmd = newcmd;
            return true;
        }
    }
    return false;
}

void adjustThrust() {
    //Mapping from +- 100% thrust/cwturn to PWM signals
```

---

```

// to motor. Need to be tested
    int newLeftMotor = 1.5*cmd.thrust + cmd.ccwturn;
    int newRightMotor = 1.5*cmd.thrust - cmd.ccwturn;

if(newLeftMotor > leftMotor + dm){
    leftMotor += dm;
} else if(newLeftMotor < leftMotor - dm){
    leftMotor -= dm;
} else {
    leftMotor = newLeftMotor;
}

    if (leftMotor > maxout) {
        analogWrite(leftMotorForward, 255);
        analogWrite(leftMotorBackward, 0);
    } else if (leftMotor > 0) {
        analogWrite(leftMotorForward, leftMotor+
deadband);
        analogWrite(leftMotorBackward, 0);
    } else if(leftMotor < -maxout) {
        analogWrite(leftMotorForward, 0);
        analogWrite(leftMotorBackward, 255);
    } else if(leftMotor < 0) {
        analogWrite(leftMotorForward, 0);
        analogWrite(leftMotorBackward, -leftMotor +
deadband);
    } else {
        analogWrite(leftMotorForward, 0);
        analogWrite(leftMotorBackward, 0);
    }
}

if(newRightMotor > rightMotor + dm){
    rightMotor += dm;
} else if(newRightMotor < rightMotor - dm){
    rightMotor -= dm;
} else {
    rightMotor = newRightMotor;
}

    if (rightMotor > maxout) {
        analogWrite(rightMotorForward, 255);
        analogWrite(rightMotorBackward, 0);
    }else if (rightMotor > 0) {
        analogWrite(rightMotorForward, rightMotor +
deadband);

```

---

```

        analogWrite(rightMotorBackward, 0);
    } else if(rightMotor < -maxout) {
        analogWrite(rightMotorForward, 0);
        analogWrite(rightMotorBackward, 255);
    } else if(rightMotor < 0) {
        analogWrite(rightMotorForward, 0);
        analogWrite(rightMotorBackward, -rightMotor +
deadband);
    } else {
        analogWrite(rightMotorForward, 0);
        analogWrite(rightMotorBackward, 0);
    }
}

void sendPushingForce(){
    push.force = scale.get_units(1);

    radio.stopListening();
    bool ok = radio.write( &push, sizeof(
    tugboat_control::PushingForce) );
    if(ok)
    {
        OK(true);
    }
    else
    {
        OK(false);
    }
    radio.startListening();
}

void OK(bool isOK){
    if(isOK){
        digitalWrite(OK_LED, HIGH);
        digitalWrite(NOT_OK_LED, LOW);
    }
    else {
        digitalWrite(OK_LED, LOW);
        digitalWrite(NOT_OK_LED, HIGH);
    }
}

void setup ()
{
    Serial.begin(57600);

```

---



---

```

pinMode(OK_LED, OUTPUT);
pinMode(NOT_OK_LED, OUTPUT);
OK(false);

radio.begin();
radio.setRetries(15,15);
radio.openWritingPipe(txPipe);
radio.openReadingPipe(1,rxPipe);
radio.startListening();

scale.set_scale(SCALE);
// this value is obtained by calibrating the scale
// with known weights; see the README for details
scale.tare(); // reset the scale to 0
scale.power_up();

pinMode(leftMotorForward, OUTPUT);
pinMode(leftMotorBackward, OUTPUT);
pinMode(rightMotorForward, OUTPUT);
pinMode(rightMotorBackward, OUTPUT);

cmd.ID = TUG_ID;
cmd.thrust = 0;
cmd.ccwturn = 0;

push.ID = TUG_ID;
push.force = 0;
adjustThrust(); // init with all motors still

OK(true);

while(!getNewCommand())
{ //Reset scale when first command is received
  scale.tare();
}
}

void loop()
{
  unsigned long start = millis();
  if(getNewCommand()){
    sendPushingForce();
    OK(true);
  }
}

```

---

```
else
{
    OK(false);
}
adjustThrust();
unsigned long delayTime = millis() - start;
if(delayTime < 100){
    delay(delayTime);
}
}
```

---

```

#ifdef _PID_H_
#define _PID_H_

class PIDImpl;
class PID
{
    public:
        // Kp - proportional gain
        // Ki - Integral gain
        // Kd - derivative gain
        // dt - loop interval time
        // max - maximum value of manipulated variable
        // min - minimum value of manipulated variable
        PID( long dt, double max, double min, double Kp,
            double Kd, double Ki , bool isCircular);

        // Returns the manipulated variable given a
        // setpoint and current process value
        double calculate( double setpoint, double pv );
        ~PID();

        //Resets integral effect, derivate effect and
        // timechange
        void resetPID();

    private:
        PIDImpl *pimpl;
};

//Normalizes angle to +- pi radians
double PIDnormalizeAngle(double angle);

long PIDmsNow();

#endif

```

---

```

#ifdef _PID_SOURCE_
#define _PID_SOURCE_

#include <iostream>
#include <cmath>
#include <chrono>
#include "pid.h"

using namespace std;

class PIDImpl
{
public:
    PIDImpl( long dt, double max, double min,
            double Kp, double Kd, double Ki, bool
            isCircular );
    ~PIDImpl();
    double calculate( double setpoint, double pv );
    void resetPID();

private:
    long _dt;
    double _max;
    double _min;
    double _Kp;
    double _Kd;
    double _Ki;
    bool _isCircular;
    double _output;
    double _pre_error;
    double _pre_Dout;
    double _integral;
    long _last_time;
    bool _isReset;
};

PID::PID( long dt, double max, double min,
          double Kp, double Kd, double Ki, bool isCircular )
{
    pimpl = new PIDImpl( dt, max, min, Kp, Kd, Ki, isCircular );
}
double PID::calculate( double setpoint, double pv )
{
    return pimpl->calculate( setpoint, pv );
}

```

---

```

}
void PID::resetPID()
{
    pimpl->resetPID();
}
PID::~~PID()
{
    delete pimpl;
}

long PIDmsNow()
{
    long now = (long)std::chrono::duration_cast<
        std::chrono::milliseconds>(std::chrono::
            system_clock::now().time_since_epoch())
        .count();
    return now;
}

double PIDnormalizeAngle(double angle)
{
    if(angle > 3.14)
    {
        angle -= 6.28;
    }
    else if(angle < -3.14)
    {
        angle += 6.28;
    }
    return angle;
}

/**
 * Implementation
 */
PIDImpl::PIDImpl( long dt, double max, double min,
    double Kp, double Kd, double Ki, bool isCircular ) :
    _dt(dt),
    _max(max),
    _min(min),
    _Kp(Kp),
    _Kd(Kd),
    _Ki(Ki),

```

---

```

    _isCircular(isCircular),
    _output(0),
    _pre_error(0),
    _pre_Dout(0),
    _integral(0),
    _last_time(PIDmsNow())
{
}

double PIDImpl::calculate( double setpoint, double pv )
{
    long this_time = PIDmsNow();
    double timechange = (double)(this_time - _last_time)
        / 1000.0;
    if(timechange > _dt){ //else return previous output
        _last_time = this_time;
        // Calculate error
        double error = setpoint - pv;

        //Fix if circular error crosses +/- pi
        if(_isCircular)
        {
            error = PIDnormalizeAngle(error);
        }

        // Proportional term
        double Pout = _Kp * error;

        if(_isReset)
        {
            _output = Pout;
            _isReset = false;
        }
        else
        {
            _integral += error * timechange;
            double Iout = _Ki * _integral;

            // Derivative term
            double dError = error - _pre_error;
            if(_isCircular)
            {
                dError = PIDnormalizeAngle(dError);
            }
            double Dout = _pre_Dout * 0.7 + 0.3 *

```

---

```

        _Kd * dError / timechange;
        _pre_Dout = Dout;

        _output = Pout + Iout + Dout;
    }

    // Save error to previous error
    _pre_error = error;
    // Restrict to max/min
    if( _output > _max )
    {
        _output = _max;
    }
    else if( _output < _min )
    {
        _output = _min;
    }
}

if(isnan(_output)){
    _output = 0;
}
return _output;
}

PIDImpl::~PIDImpl()
{
}

void PIDImpl::resetPID()
{
    _output = 0;
    _integral = 0;
    _pre_Dout = 0;
    _isReset = true;
    std::cout << "Resetting PID\n";
}

#endif

```





Appendix **B**

Risk assessment

NTNU		Utarbeidet av		Nummer		Dato	
		HMS-avd.		HMSRVZ601		22.03.2011	
HMAS		Godkjent av		Erstatter		01.12.2006	
		Rektor					
<b>Kartlegging av risikofylt aktivitet</b>							

Enhet: IPM

Dato: 03.02.2017

Linjeleder: Torgeir Velo

Delakere ved kartleggingen (m/ funksjon): Sondre Naess Midtskogen (student), Martin Steinert (veileder)

(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave Sondre Naess Midtskogen. Ship Control by Autonomous Tugboats.

Er oppgaven rent teoretisk? (JA/NEI): NEI «JA» betyr at veileder innestår for at oppgaven ikke inneholder noen aktiviteter som krever

risikourdering. Dersom «JA»: Beskriv kort aktiviteten i kartleggingskjemaet under. Risikourdering trenger ikke å fylles ut.

Signaturer: Ansvarlig veileder:



Student: Sondre Naess

ID nr.	Aktivitet/prosess	Ansvarlig	Eksisterende dokumentasjon	Eksisterende sikringsiltak	Lov, forskrift o.l.	Kommentar
1	Lodding			Avtrekk ved loddestasjon, førstehjelpsutstyr på verkstedet		
2	Laserkutter			Oppsyn av laserkutter under bruk, brannslukningsapparat i umiddelbar nærhet		
3						
4						
5						
6						



NTNU				Risikovurdering	
HMS				Utarbejdet av HMS-avd. Godkjent av HMSRRV2601 Erstatler Rektor	Nummer HMSRRV2601
					

## Sannsynlighet vurderes etter følgende kriterier:

Svært liten 1	Liten 2	Middels 3	Stor 4	Svært stor 5
1 gang pr 50 år eller sjeldnere	1 gang pr 10 år eller sjeldnere	1 gang pr år eller sjeldnere	1 gang pr måned eller sjeldnere	Skjer ukentlig

## Konsekvens vurderes etter følgende kriterier:

Gradering	Menneske	Ytre miljø Vann, jord og luft	Øk/materiell	Omdømme
<b>E</b> Svært Alvorlig	Død	Svært langvarig og ikke reversibel skade	Drifts- eller aktivitetsstans > 1 år.	Troverdighet og respekt betydelig og varig svekket
<b>D</b> Alvorlig	Alvorlig personskade. Mulig uførtet.	Langvarig skade. Lang resitusionsstid	Driftsstans > ½ år Aktivitetsstans i opp til 1 år	Troverdighet og respekt betydelig svekket
<b>C</b> Moderat	Alvorlig personskade.	Mindre skade og lang resitusionsstid	Drifts- eller aktivitetsstans < 1 mnd	Troverdighet og respekt svekket
<b>B</b> Liten	Skade som krever medisinsk behandling	Mindre skade og kort resitusionsstid	Drifts- eller aktivitetsstans < 1uke	Negativ påvirkning på troverdighet og respekt
<b>A</b> Svært liten	Skade som krever førstehjelp	Ubetydelig skade og kort resitusionsstid	Drifts- eller aktivitetsstans < 1dag	Liten påvirkning på troverdighet og respekt

## Risikoverdi = Sannsynlighet x Konsekvens

Beregn risikoverdi for Menneske. Enheten vurderer selv om de i tillegg vil beregne risikoverdi for Ytre miljø, Økonomimateriell og Omdømme. I så fall beregnes disse hver for seg.

## Til kolonnen "Kommentarer/status, forslag til forebyggende og korrigerende tiltak":

Tiltak kan påvirke både sannsynlighet og konsekvens. Prioriter tiltak som kan forhindre at hendelsen inntreffer, dvs. sannsynlighetsreduserende tiltak foran skjerpet beredskap, dvs. konsekvensreduserende tiltak.

NTNU		Risikomatrise		utarbeidet av		Nummer		Dato	
				HMS-avd.		HMSRV2604		08.03.2010	
HMS/IS				godkjent av				Erstatter	
				Rektor				09.02.2010	
									

## MATRISE FOR RISIKOVURDERINGER ved NTNU

<b>KONSEKVENNS</b>					
Svært alvorlig	E1	E2	E3	E4	E5
Alvorlig	D1	D2	D3	D4	D5
Moderat	C1	C2	C3	C4	C5
Liten	B1	B2	B3	B4	B5
Svært liten	A1	A2	A3	A4	A5
	Svært liten	Liten	Middels	Stor	Svært stor
<b>SANNSYNLIGHET</b>					

Prinsipp over akseptkriterium. Forklaring av fargene som er brukt i risikomatrisen.

Farge	Beskrivelse
Rød	Uakseptabel risiko. Tiltak skal gjennomføres for å redusere risikoen.
Gul	Vurderingsområde. Tiltak skal vurderes.
Grønn	Akseptabel risiko. Tiltak kan vurderes ut fra andre hensyn.



Appendix **C**

Pre-master thesis

# Automated ship docking using autonomous swarm tugboats: A preliminary concept study

Sondre Næss Midtskogen

December 2016

Project thesis

Department of Product Development and Materials  
Norwegian University of Science and Technology

Supervisor: Professor Martin Steinert

Co-supervisor: Kristoffer B. Slåttsveen





**PROJECT WORK FALL 2016  
FOR  
STUD.TECHN. Sondre Midtskogen**

**Autonomous tugboat operations**

How to conduct tugboat operations with zero human interaction. Conceptualize, test and scale.

- Generate concepts
- Build prototypes
- Build test setups
- Test and compare alternatives
- Judge and evaluate concepts

Also, it is expected to contribute to one or more scientific publications during the project/master thesis.

The supporting coach Kristoffer Slåttsveen.

**Formal requirements:**

Students are required to submit an A3 page describing the planned work three weeks after the project start as a pdf-file via "IPM DropIT" (<http://129.241.88.67:8080/Default.aspx>). A template can be found on IPM's web-page (<https://www.ntnu.edu/ipm/project-and-specialization>).

Performing a risk assessment is mandatory for any experimental work. Known main activities must be risk assessed before they start, and the form must be handed in within 3 weeks after you receive the problem text. The form must be signed by your supervisor. Risk assessment is an ongoing activity, and must be carried out before starting any activity that might cause injuries or damage materials/equipment or the external environment. Copies of the signed risk assessments have to be put in the appendix of the project report.

No later than 1 week before the deadline of the final project report, you are required to submit an updated A3 page summarizing and illustrating the results obtained in the project work.

Official deadline for the delivery of the report is 13 December 2016 at 2 p.m. The final report has to be delivered at the Department's reception (1 paper version) and via "IPM DropIT".

When evaluating the project, we take into consideration how clearly the problem is presented, the thoroughness of the report, and to which extent the student gives an independent presentation of the topic using his/her own assessments.

The report must include the signed problem text, and be written as a scientific report with summary of important findings, conclusion, literature references, table of contents, etc.

Specific problems to be addressed in the project are to be stated in the beginning of the report and briefly discussed. Generally the report should not exceed thirty pages including illustrations and sketches.

Additional tables, drawings, detailed sketches, photographs, etc. can be included in an appendix at the end of the thirty page report. References to the appendix must be specified. The report should be presented so that it can be fully understood without referencing the Appendix. Figures and tables must be presented with explanations. Literature references should be indicated by means of a number in brackets in the text, and each reference should be further specified at the end of the report in a reference list. References should be specified with name of author(s) and book, title and year of publication, and page number.

Contact persons:

At the department	Martin Steinert, Kristoffer Slåtsveen
From the industry	DNA



Martin Steinert  
Supervisor



**NTNU**  
Norges teknisk-  
naturvitenskapelige universitet  
Institutt for produktutvikling  
og materialer

## Summary

The project assignment defined at start of term is a fuzzy front end development challenge on the topic “Autonomous tugboat operations”. After doing a quick market analysis, in-harbour ship guidance and docking assistance was identified as a promising subset of tugboat operations to autonomize. The following research problems were defined:

1. Clearly, some sort of artificial intelligence is required to control the tugboats. I need to learn about the problem of vessel control through tugboat control, identify the tools I can use to solve it, and learn enough about them to be able to develop and implement an AI.
2. How can I use learning prototypes to help with research problem 1?
3. I will need a physical prototype setup on which to implement the AI. What information is ultimately needed, and how should this information flow between the different agents? Which sensors, actuators and other components do I need to represent this system on a scale setup? Design and begin development of this prototype setup.

Through the course of the project, I have studied control theory in depth, and machine learning to a lesser extent, to address research problem 1. I have made a programming environment for rapid prototyping of tugboat control algorithms, to address problem 2, and I have designed and started development of a physical setup for validation testing of control schemes, as described by research problem 3.

The project results are subject knowledge and tools, plans for the physical test setup, and the following set of potential research questions for my master thesis:

1. Build a physical test setup for tugboat control, as described in chapter 6.
2. Implement different control schemes in the test setup, based on the literature discussed in 4.5, and evaluate them in terms of reliability and efficiency.
3. Develop a new control scheme with dynamic tugboat positioning, implemented as a state machine or otherwise, using the ship’s main thruster when appropriate.
4. Make recommendations on size, number and control of tugboats for large-scale implementation.

My specialization course has been TMM4280 Advanced Product Development.

# Contents

1	Introduction .....	1
1.1	The challenge .....	1
1.2	Topic introduction .....	1
1.2.1	Tugboats.....	1
1.2.2	Unmanned tugboat in hazardous environments.....	2
1.2.3	Autonomous vehicles and swarm technology.....	2
1.3	Project introduction.....	3
1.3.1	Scope of pre-master .....	3
1.3.2	Limitations .....	3
1.3.3	Problem definition .....	3
1.3.4	Identify research questions .....	4
1.3.5	Outline.....	4
2	Method.....	5
2.1	My approach.....	5
2.2	Prototyping .....	5
2.2.1	Set-based prototype.....	6
3	Set-based digital prototype for tugboat control prototyping .....	8
3.1	Setup.....	8
3.2	Learnings .....	8
3.2.1	“Out of the way!” algorithm .....	9
3.2.2	“Protective circle” algorithm .....	10
3.3	Further use.....	11
4	Deep-dive learning on the topic of ship control by tugboat swarm.....	12
4.1	Problem definition.....	12
4.2	Tugboat control by machine learning.....	12

4.3	Tugboat control by control theory.....	13
4.4	Adaptive control: The best of both worlds.....	14
4.5	Autonomous barge control and ship docking in literature .....	16
4.6	Limitations of current research and potential for further work.....	18
5	Physical test setup.....	20
5.1	General requirements .....	20
5.2	Design.....	20
5.2.1	Drones .....	20
5.2.2	Instrumentation .....	21
5.2.3	Information flow- and handling .....	22
5.3	Progress at project finish .....	22
6	Preliminary research problems for master thesis.....	24
7	Appendices .....	27

## **Appendices**

- A. Code for set-based prototype in Processing IDE
- B. Example code used as basis for computer vision
- C. Risk assessment

# 1 Introduction

## 1.1 The challenge

The project assignment defined at start of term is a fuzzy front end development challenge on the topic “Autonomous tugboat operations”. The assignment is open-ended, but with the expectation that I bring something of value to the project sponsor, Seatex. Part of the assignment will be to figure out what I will focus on.

## 1.2 Topic introduction

### 1.2.1 Tugboats

Tugboats main purpose is to help large ships with limited manoeuvrability, or barges with no power of their own, move in and out of ports and dock in tight harbour settings. They control the movement of their charge by pushing against its hull or by pulling on tug lines, often working together in teams of several tugboats. The tugboats are operated by crews of highly skilled professionals with years of experience, relying on radio communication and procedure familiarity to coordinate their actions. Modern tugboats also perform other tasks, such as escort, firefighting and pollution control.

According to a market report from 2007 [1], the market’s need for tugboats is growing. Almost half of all newbuildings are harbour tugboats, which are smaller, more compact and more nimble than seagoing tugs. The trend in the market is towards bigger and better “tractor tugs”, capable of performing the work of two or three traditional tugboats. *“Older tugs were designed to operate with five to eight man crews. Depending on the trade, tugs today are being designed to operate with anywhere from two to six crew. This is necessary, not only as a cost savings, but because of the extreme shortage of qualified personnel. While I do not expect to see an “Unmanned Tug” in my lifetime, more automation is required, not only in the engineroom, but on deck.”*

While we might be years away from fully replacing tugboat crew with computers, most of their time is spent guiding ships into the harbour and assisting with docking. This is a routine task that can be fully automated, and will therefore be the focus of my work.

### **1.2.2 Unmanned tugboat in hazardous environments**

Several actors are working on improving crew safety by introducing unmanned surface vehicles in the more dangerous parts of tugboat operations. Canadian marine engineering company Robert Allan Ltd. are developing a line of remotely operated tugboats called RAmora [2]. With no steering house or other crew facilities, it is more compact than tugboats of similar performance. It has an automatic towline and firefighting capabilities, and can operate in hazardous environments unsuitable for humans. RAmora is controlled by a professional tugboat captain from a “command tug” a safe distance away, through advanced telepresence technology.

CART is an automatic towline attachment system developed by Italian Posidonia [3]. It removes the human element from the high-risk operation of attaching a towline to the towing system of distressed ships, by using a small USV on a leash. The USV swims around a buoy launched from the distressed vessel, tying the towline to the buoy in the process.

While I will not be focusing on these types of problem, these actors show that the tugboat market is willing to apply automated technology.

### **1.2.3 Autonomous vehicles and swarm technology**

We are in the middle of an autonomous revolution – autonomous vehicles are taking over the roads, the skies and the seas. Autonomous vehicles are a hot research topic, both in academia and the industry. For example, six out of Kongsberg’s nine student summer projects in 2016, projects that focus on emerging technologies, were on autonomous vehicles.

Like with tugboats, the trend with all vehicles in the past decades, has been to build bigger and bigger, while maintaining a similar crew size. In an ever more competitive world, maximum capacity per crew has been the optimum. But with autonomization, this is no longer a concern. The first autonomous buses to come to Norway have seating for six, and can take a total of 12 passengers [4], offering much more adaptability than the current large city buses. While building several small is still more expensive than one big, the new size sweet spot is decided by the need for adaptability, opening up for downsizing and multiplication, up to robot swarm levels.

A swarm of robots is categorized as a large group of relatively simple robots, working together through local communication, to complete a task that is beyond the capabilities of any of its individuals [5]. I do not yet know to which degree I will adhere to the concept of



swarm robotics. For the remainder of this paper, I will use the word “swarm” to mean a group of more than three robots working together.

## **1.3 Project introduction**

### **1.3.1 Scope of pre-master**

The whole project is divided in two parts, with the pre-master project constituting one third of the workload (as measured in ECTS). The goal of the pre-master project is to define the assignment and explore the solution space in order to get a good starting point for the master thesis. This is done by rigorously conceptualising, testing and validating solutions, identifying points of interest and potential pitfalls, and generally learning as much as possible about the topic. The pre-master project spans the fuzziest phase of the project, from loosely defined topic to a concise set of research problems and a plan of action for the master thesis.

### **1.3.2 Limitations**

Some limitations are defined to narrow down the scope of the project.

- While tugboats are used for a range of tasks, I will focus on ship docking and navigation assistance.
- Despite the “tug” in “tugboat”, I will only consider vessel guidance by pushing – designing an automated tug line deployment system would be a sizeable project onto itself.
- I will not concern myself with any higher-level operations such as harbour logistics, route planning or obstacle avoidance. My aim is to be able to make a given ship move where and how I want, using autonomous tugboats.
- I will not research, defer to, or try to change any rules or regulations regarding autonomous surface vehicles or maritime activities in general. This is purely a technology push.

### **1.3.3 Problem definition**

Consider the following case. *A large ship arrives outside a harbour, intending to dock. The ship is only actuated by one or more rear thrusters and rudders, which is insufficient for the fine manoeuvring necessary to dock safely, and requires assistance.*

Through the course of my project and master thesis, I will develop a concept for providing this assistance using a swarm of autonomous tugboats, and realize the concept on a scale setup.

### **1.3.4 Identify research questions**

The very first thing I do is arrange a brainstorming session within the TrollLABS community on the topic of autonomous tugboat operations. Working alone, my capacity for divergent thinking is limited, and project kick-off is the time at which divergent thinking is most critical. The purpose of the session is to start my project with a broad view of the problem, influenced by multiple input angles, have my own thoughts on the topic externally evaluated, and ultimately identify interesting problems to address. Three initial research problems are defined:

1. Clearly, some sort of artificial intelligence is required to control the tugboats. I need to learn about the problem of vessel control through tugboat control, identify the tools I can use to solve it, and learn enough about them to be able to develop and implement an AI.
2. How can I use learning prototypes to help with research problem 1?
3. I will need a physical prototype setup on which to implement the AI. What information is ultimately needed, and how should this information flow between the different agents? Which sensors, actuators and other components do I need to represent this system on a scale setup? Design and begin development of this prototype setup.

### **1.3.5 Outline**

While deeply interconnected, the research problems described above are best treated as separate problems, for the purpose of presentation. In section 2, I will describe the methods used to address each research problem. In sections 3 through 5, I will describe the steps taken to solve each problem, and present results in terms of learning outcome and further use in my master thesis. In section 6, I will formulate suitable research problems and a plan of execution for my master thesis, based on the results of sections 3 through 5.

## 2 Method

### 2.1 My approach

To the project at large, I will take a wayfaring approach [6], and use the different PD tools in my belt to find my way. Wayfaring is an analogy to PD efforts where the end goal – and the way there – is initially unknown. Rather than planning the whole journey from start to finish, like in more traditional engineering design, the route is planned incrementally, each step selected based on the current most promising direction. As the challenge is open-ended, and the topic unfamiliar to me, this approach is favourable. The course of my work will be a succession of questions and answers, where each set of answers provides a new set of questions. With limited time and resources, it is crucial that I maximise my learning to effort rate. I will therefore not pursue any single concept or subject in depth. Instead, I will focus on identifying things that are useful to my project, and build a broad foundation on which to base my master thesis.

In parallel with this, I will take a deep dive into control theory by taking two core cybernetic courses – *Linear system theory* and *Nonlinear control systems* – as well as study basic machine learning, as these are the two most important tools for research problem 1.

### 2.2 Prototyping

At the Department of Engineering Design at NTNU, we believe strongly in using prototyping as a learning tool. But how exactly do I go about prototyping autonomous tugboat operations, i.e. research problem 2? The classical approach is to break the problem down to its parts, figure out each part separately, and put the parts together. This way, I must build a swarm of radio controlled boats, control them using some form of position tracking, and gradually implement intelligent control. I will do this at some point (this is research problem 3), using the described method, but it will take several months at the very best, before this setup is up and running to the point where I can begin prototyping control algorithms. And even then, running and maintaining an experiment setup like this will be a lot of work, and progress will inevitably be slow.

An experience prototype was considered, in which a group of people each controlled one RC tugboats, working together to push a surface vessel in a controlled manner. As my coach quaintly put it; “Prototyping AI using I”. This was discarded as a bad time investment, and

would likely provide less insight than a 5 minute YouTube video on harbour tugboats, in addition to having all the drawbacks of the “divide and conquer”-method.

Instead, I will make a *set-based* prototype with which I can explore and familiarize myself with tugboat control algorithms, using the physical prototype setup as a validation platform for the most promising control algorithms.

As this is a technology push-assignment, human interaction is not considered – all prototypes are function prototypes [7].

### **2.2.1 Set-based prototype**

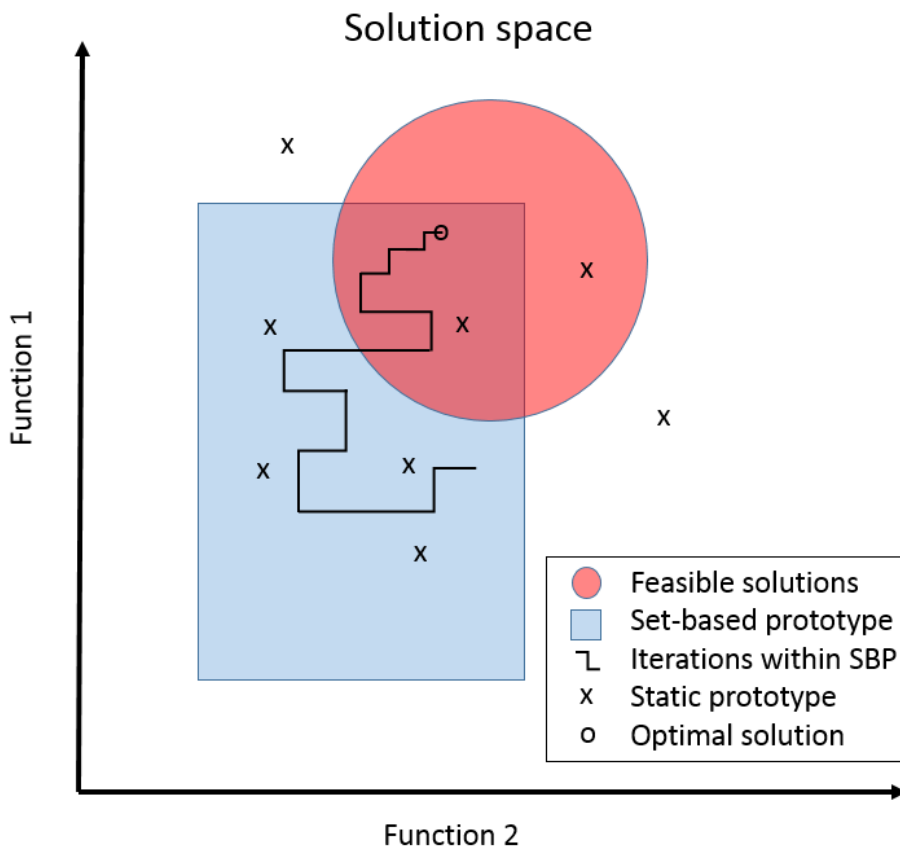
The term “set-based prototype” is coined by Christer Elverum [8], and is still under development. The idea is to identify the main decision variables of your problem, and make a dynamic prototype in which these are adjustable. This way, one can address the overall functionality of a product, and the interplay between the main decision variables, rather than specific functions or concepts that can be isolated and studied on their own. It is a top-down approach, rather than the normal bottom-up.

If the solution space is an  $n$ -dimensional Cartesian space in which all possible design solutions can be found, static prototypes represent points in this space. As engineers, we excel at informed guesses on what may or may not work, and we use this skill when designing a validation prototype – in other words, where in the design space we place the prototype. This is fine when exploring simple problems in low-dimensional subset of the solution space, but becomes increasingly difficult with more complex problems. For comparison, finding the minimum of a well-defined one-dimensional function is as simple as setting the derivative equal to zero, and compare the resulting values. But once a few more dimensions and some complexity is added, the problem becomes difficult enough to warrant a whole branch of mathematics – optimization. In mathematical optimization, finding a feasible point in which to start your optimization algorithm is not always trivial. In the Simplex method [9], an algorithm for linear programming, an optimization problem of the same dimension as the original problem is solved in order to find a basic feasible starting point. An engineer’s best guess is no longer good enough.

A set-based prototype represents a regular  $n$ -dimensional polytope in the solution space, where  $n$  is the number of decision variables. By adjusting the parameters of the prototype in a manner similar to the *steepest descent algorithm* in optimization (i.e. which changes will make the biggest improvement), one is not only much more likely to find a feasible solution

than with a static prototype; the dynamics between the decision variables are more apparent, and in the case where no feasible solution exists, this can be claimed with greater certainty. The difference between a static and a set-based prototype, with  $n = 2$ , is illustrated in Figure 1.

The method is especially effective in the uncertain early phases, when exploring multiple overall concepts based on known feasible sub-concepts. However, a set-based prototype is by necessity of high complexity, and is therefore much more resource intensive than a simple static prototype.



**Figure 1:** Static vs. Set-based prototype

## **3 Set-based digital prototype for tugboat control prototyping**

### **3.1 Setup**

I define my main decision variables to be the number of tugboats, tugboat size and thrust strength, and the whole process for tugboat control. This includes what information is fed into the control algorithm, if and how this information is shared between tugboats, and everything that happens before this is translated into output thrust. I also make a variable current to test the algorithms under different conditions. I assume all other aspects of the tugboats to be by-products of these decisions, and can be approximated by any generic boat. Other parameters, such as water viscosity, conservation of energy, Newton's laws of motion etc. are constant environment parameters.

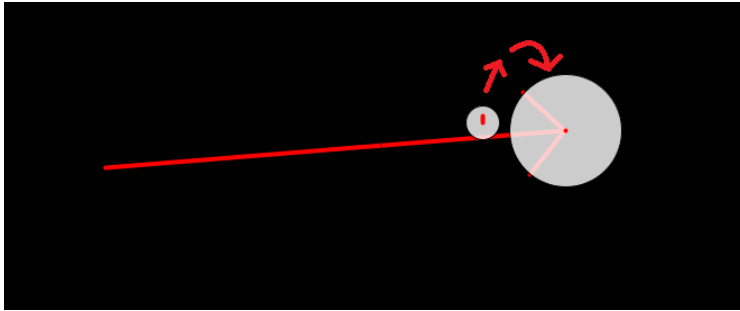
I use a programming environment called Processing to make a computer simulated set-based prototype. Processing is a Java-based IDE, designed for simple visual programming. It is the IDE on which the Arduino IDE is based, and like the Arduino IDE it has a large toolbox of premade functions, classes and example sketches. To save work, I use an example sketch with 2D bouncing balls as a base. Tugboats are modelled as small balls, the ship is modelled as a large ball, and the remaining decision variables and environment parameters are implemented in the simulation. I now have an environment in which I can quickly and easily alter the decision variables in my prototype by changing a few lines of code, and experiment setup time is reduced to a mouse click. The current version of the program code can be found in Appendix A.

### **3.2 Learnings**

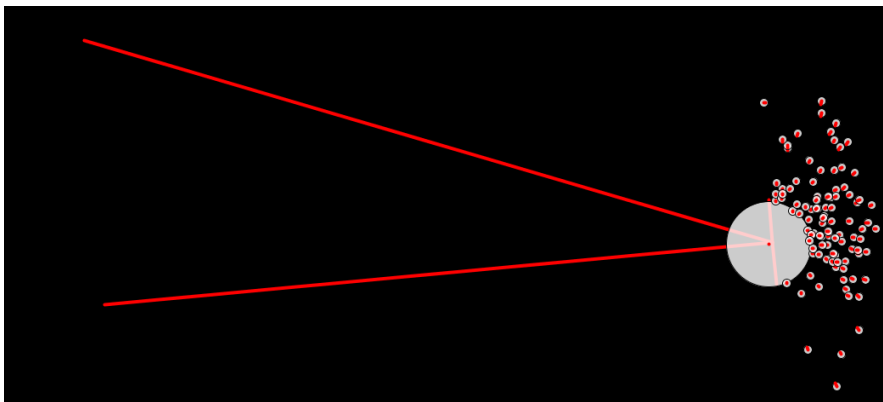
I have not made an exhaustive study of control algorithms using this tool, nor does the simulation represent a physical perfectly. That was never the intention. Instead, these have been the digital equivalents to paper prototypes – a way for me to quickly gain an understanding of the problem, what works and what does not. Let me present two of my early control algorithms to illustrate.

### 3.2.1 “Out of the way!” algorithm

The “Out of the way!” algorithm is very simple. All tugboats try to get to the centre of the ship thus pushing against its hull, except if they are “in the way”, in which case they move tangential to the ship’s hull until they are no longer in the way. A tugboat is in the way if it is within a certain angle from the ship’s desired direction, see figures 2 and 3.



**Figure 2:** “Out of the way!” algorithm with 1 tugboat. The long red line marks the ship’s desired force vector, the two short red lines mark the “In the way” zone, and the red arrows marks the tugboat’s future trajectory.

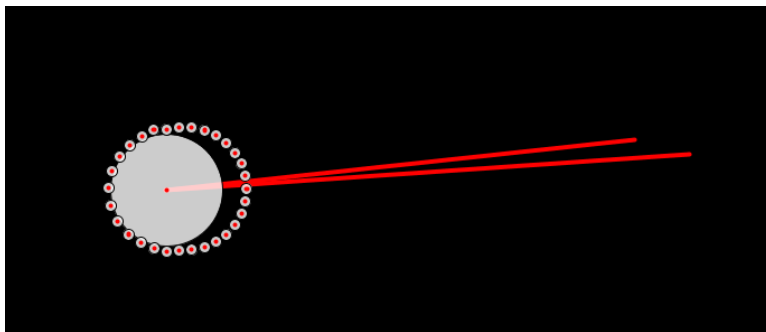


**Figure 3:** “Out of the way!” algorithm with 100 tugboats. The top red line is the resultant force provided by the tugboats, captured at a presentable time.

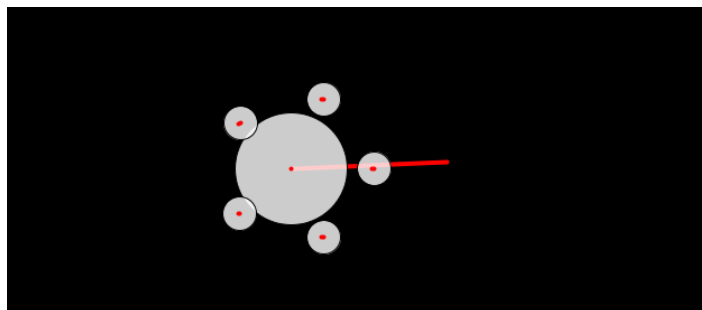
This algorithm works better with many small tugboats, but lacks a stopping mechanism. When desired force is set to zero, the tugboats will keep pushing towards the middle, but can do so from every angle. However, as they are all bunched up on one side after a push, they will just keep pushing the ship in the same direction. A more defensive approach is called for.

### 3.2.2 “Protective circle” algorithm

In the “Protective circle” algorithm, the tugboats are evenly spaced around the ship’s perimeter, and are told to hold that relative position using two PD (Proportional-Derivative, like a mass-spring-damper system) controllers, one for each axis. When a desired force vector is applied, each tugboat exerts a thrust in proportion to this vector, moving slightly away from their relative position, see figure 4.



**Figure 4:** “Protective circle” algorithm with 30 tugboats. See how the majority of the tugboats are not pushing against the ship.



**Figure 5:** “Protective circle” algorithm with 5 tugboats, and desired force set to zero. The tugboats are breaking the leftward velocity of the ship.

The problem with this algorithm is that at any time, most tugboats will not be pushing the ship in the right direction, see figure 4. However, positioning works very well, and as a bonus, the algorithm inadvertently inherits a breaking mechanism. In figure 5, the tugboats are exerting a rightwards force on the ship, even though the desired force is set to zero. This is because the ship has leftward velocity, and the tugboats seek to hold a stationary perimeter around the ship.



### **3.3 Further use**

The simulation has every potential for refinement, and can be upgraded to a higher-resolution prototype. By implementing a noncircular ship and a more realistic representation of the tugboat thrusters, I can use this environment to benchmark control algorithms and weed out bugs, before implementing them on the physical setup. This will likely save me a lot of time in the long run.

The environment can also be used to simulate swarm-based control algorithms, with very large number of tugboats. While not in my main line of study, this is something I want to study on the side, if I have the time.

## 4 Deep-dive learning on the topic of ship control by tugboat swarm

### 4.1 Problem definition

The most fundamental problem on the topic of ship control by tugboat swarm can be defined in the following way:

*By the combined efforts of a swarm of tugboats able to exert force on a surface vessel by pushing against its hull, the vessel's inertial position, velocity and acceleration in the surface plane should be brought to the desired values, within an acceptable margin of error after an acceptable amount of time.*

All higher-level problems assume this to be true, for some margin of error and some amount of time.

### 4.2 Tugboat control by machine learning

Of the three categories of machine learning – supervised, unsupervised and reinforcement learning – the latter is the most useful for this problem. Reinforcement learning algorithms can be provided with a simple high-level objective, like minimizing the distance between the desired position of the ship and the actual position, formulated as a fitness function. Over time, the algorithm changes to improve its fitness, either by some self-adaptation algorithm or in the case of evolutionary algorithms, by “breeding” with other algorithms.

Reinforcement machine learning algorithms can be trained to solve a problem without being explicitly told how to do it, and can sometimes outperform deterministic algorithms.

Algorithms have learned how to play Super Mario [10] and Mario Kart [11], build roller coasters [12] and even hunt food, avoid predators and fight rivals [13]. In a similar manner, I can train tugboats to push a ship to a given position.

The algorithms are trial and error-driven. They keep behaviour that gives better performance, and discards behaviour that does not. The key to developing a reinforcement learning algorithm is to present the fitness function in a way that gives a direct correlation between input, output and performance. If one exists, this method will eventually find this correlation. It is, however, a very time-consuming process.

For my problem, most actions will, under most circumstances, not affect the distance between the desired and actual position of the ship. The likelihood of my tugboat learning to perform the sequence of actions required to positively affect its fitness function is very small, and the algorithm will not learn anything. However, if I expand the fitness function to include minimizing the distance between the tugboat and the ship, the tugboat will learn to keep close to the ship. Now, the likelihood of affecting the position of the ship is much higher, facilitating faster learning.

I can even train individual tugboats to perform specialized tasks, like players in a football team. I can have “pushers” responsive for translational movement, “turners” responsible for orientation, and “brakers” to stop the ship as it approaches its destination (To deal with problems like those in 3.2.1).

The main problem with the machine learning approach, aside from the sheer amount of time it would take, comes from the same source as its main attraction. Because I do not tell the tugboats how to accomplish their goal, I have no control over their inner workings. This means that I have no idea how they will react to an unfamiliar setting, nor can I simply add layers of control, like obstacle avoidance, on top. In a busy, tight harbour setting, this lack of control and adaptability is worrying when ferrying ships of up to half a million tonnes.

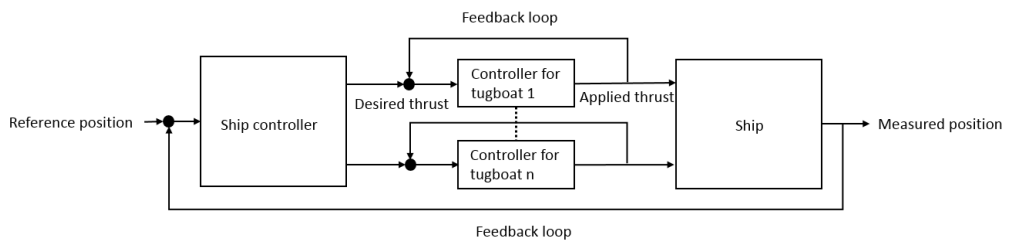
### **4.3 Tugboat control by control theory**

Unlike with machine learning, control theory requires explicit instructions on how to solve a problem. This means a lot of hard, difficult work, but also that we have full control over the inner workings of the controller. In control theory, a dynamic model of the system, called the “plant”, is formulated, and a control law is designed to drive the states of the plant to some desired value, in a desirable manner. Control theory is already being used in autopilots to keep ships on a given path, or for more heavily actuated ships with dynamic positioning capabilities, to keep them in place despite large waves and strong wind and currents.

Let us assume the position and orientation of each tugboat along the hull of the ship are known and constant, and a dynamic model of the ship is available. If so, position control of the ship, using tugboats as control output, can be reduced to a well-defined, albeit complex, over-actuated dynamic positioning system. Problems like this is addressed in existing literature such as [14].

The second assumption can be relaxed by using *robust* or *adaptive* control. Control systems usually requires a good plant model to perform well. Robust control systems are designed to be stable even with large modelling errors, which means that if I model an average ship, I can get a stable performance for a large range of ships. Adaptive control is described in detail below.

The first assumption can be fulfilled if the tugboats do exactly what I want them to, and I have perfect measurements. Neither will be the case, but if I treat the problem as two cascading systems, as shown in figure 6, I can treat the difference between expected and actual tugboat behaviour as disturbances to my system.

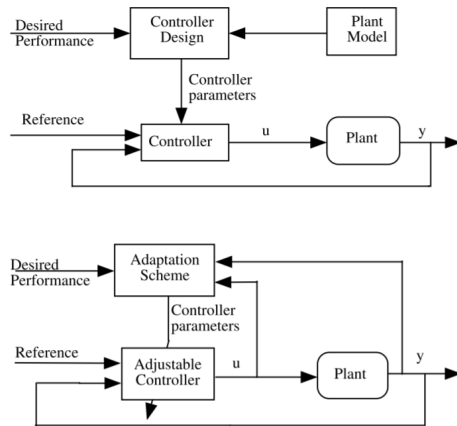


**Figure 6:** Ship control as a cascading system of two controllers

Further, it is assumed that the desired tugboat distribution is decided in advance, according to some criterion or algorithm. This is something I have to address.

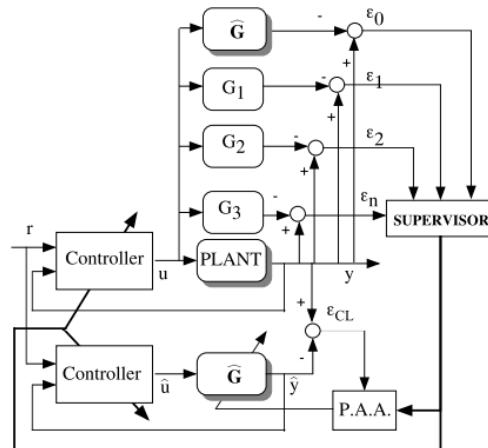
#### 4.4 Adaptive control: The best of both worlds

If all the parameters of a plant are known, an optimal controller can be designed based on the plant model and some desired performance, see figure 7 (top). However, the plant model for this problem will change a great deal from ship to ship, and explicit knowledge about the physical parameters of any given ship cannot be assumed known. To deal with this, I can use a branch of control theory called *adaptive control* [15]. In adaptive control, a controller is designed in the same way as for a static controller, based on an assumed plant model. The difference lies in that the controller changes over time in order to better achieve the desired performance, either directly or indirectly via the plant model on which the controller is based, see figure 7 (bottom).



**Figure 7:** Traditional control (top) vs. adaptive control (bottom) [15]

In adaptive control, reinforcement machine learning is used to update the controller, using the difference between the desired performance and the actual performance of the system as the fitness function. There are several different ways of implementing adaptive control, and several different algorithms to adjust controller parameters (NTNU offers a whole course on adaptive control). Finding the best implementation and adjustment algorithm is outside the scope of the pre-master project, but for a starting point I would suggest *neural net* combined with *Multiple model adaptive control with switching*, see figure 8.



**Figure 8:** Multiple model adaptive control with switching [15]

This approach is suitable for plants with large and rapid parameter variations. The method operates in two steps:

1. The best among a number of pre-defined fixed models ( $G_i$ ) is chosen for controller design.
2. The adaptive model estimator ( $\hat{G}$ ) is updated to match the best fixed model, and is continually updated using a closed-loop type parameter estimation scheme. When this model performs better than the best fixed model, it takes over controller design.

The advantage to using this method is that, for each ship, the final adaptable model can be saved to a large store of fixed models, to further improve the model convergence rate over time.

#### **4.5 Autonomous barge control and ship docking in literature**

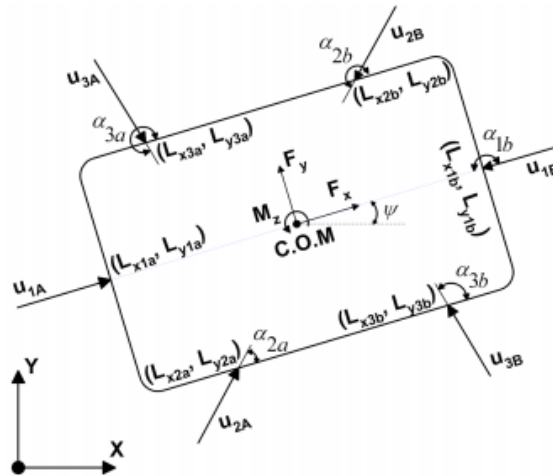
[16] developed two different control designs for a barge being controlled by 6 tugboats, with the goal of minimizing communication between tugboats. They modelled the tugboats as unidirectional positive forces at pre-defined points along the hull and constant angles of attack, see figure 9. In both cases, each tugboat had access to information about the inertial position and velocity of the barge, in addition to its own position and orientation relative to the barge centre of mass. Both control designs performed well in simulations.

In the first case, an *adaptive* controller was implemented to control the barge without any further knowledge about the model parameters. However, some limited communication between tugboats was needed to update the adaptive controller.

In the second case, no communication between the tugboats was allowed, and a *robust* control strategy was implemented. The tugboat arrangement was pre-defined and known to all tugboats, and each tugboat was programmed with the desired barge trajectory. Effects of the other tugboats were treated as disturbances to the system, and compensated for using a neural net based feedforward term.

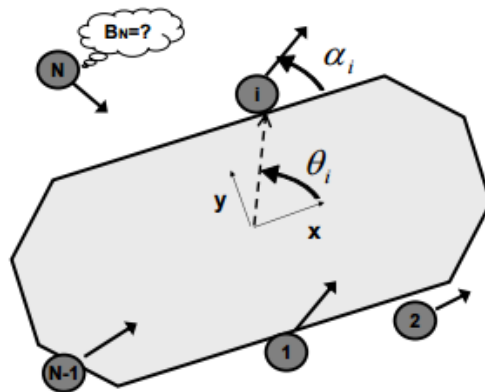
[17] proposed an algorithm for ship berthing using four autonomous tugboats, under the same rigid position assumptions as Braganza, but on the model of a ship, rather than a barge..

[18] used the same model approach as Braganza et al. to design a more robust controller, which allowed for a non-symmetric inertia matrix and perturbed tugboat positions and orientations after initiation.



**Figure 9:** Dynamic model of a barge being controlled by 6 tugboats. [16]

[19] moved away from the well-ordered, predefined distribution of 6 tugboats, in favour of a more chaotic distribution of  $N$  tugboats, thus providing a more realizable model. Assuming the tugboats are fixed to the barge and able to exert force in any direction – tugboats usually tie up to barges, and have advanced azimuth thrusters capable of 360 degrees thrust – given that the tugboats were arranged to provide force closure (the ability to exert a force or a torque in any direction), they were able to show asymptotic stability while taking the nonlinearities inherent in a physical thruster into account. Inspired by theory on multi-fingered hands [20], they formulated an optimized tugboat distribution by incrementally synthesizing planar grasp configurations, see figure 10. Two physical experiments were performed to validate these two research papers. In the first, six tugboats were represented by bilge pumps mounted to the hull of the ship. In the second, autonomous tugboats latched on to the barge using actuated magnets.



**Figure 10:** N-1 tugboats grasping a barge. Tugboat N calculates optimal attachment point. [20]

## 4.6 Limitations of current research and potential for further work

Most of the research is focused on barges, which deviates from ship in two important ways. Firstly, the assumption that tugboats are firmly attached to the controlled vessel is not always true when guiding ships. Instead, the tugboats are held in place by friction between the tugboat and the vessel's hull. This implies that the tugboats need to exert some pressure to remain in contact with the ship. Further, the control problem is in fact a two-level problem – a control algorithm is needed to hold the tugboats in place, and to provide the correct thrust as demanded by the higher-level controller, as I showed with figure 6. While the characteristics of physical thrusters [19] and the requirement of nonzero thrust [16] are both addressed, tugboat dynamics and control are not included in the analysis.

Secondly, while barges are unactuated vessels, ships have at least one powerful main thruster which can be used to propel the ship forward. [17] do not take either of these facts into account. When tugboats ferry a ship into the harbour, the ship moves mostly by its own power. The tugboats are there to keep the ship along the designated path, perform hairpin manoeuvres the ship cannot do on its own, and act as an emergency braking system. Thus, the whole team of tugboats is not needed until the final stage of ship berthing, when the tugboats take over completely.



An optimal solution could be to have a few tugboats meet the ship outside the harbour and guide it towards its destination, the ship providing its own forward momentum. Just before the ship reaches a point where additional guidance is required, a second wave of tugboats joins the efforts, and the ship powers down. The ship is then carefully guided to its destination.

While the distribution of tugboats is discussed at length, all the available research assumes a single optimal distribution, based on overall controllability. Manned tugboats frequently reposition themselves around the ship depending on the operation, for greater thrust utilization. For example, when executing a pivot, tugboats group up on one side at the stern and the other at the bow, for maximum torque. A discretely dynamic tugboat distribution can be simply implemented as a state machine, with *guided self-propulsion*, *turning clockwise*, *turning counter-clockwise* and *docking* as the different states, in addition to some safety states, and the distribution optimized for each state individually. Or it could be implemented fluidly using more advanced methods.

Another problem that is not addressed is to find the ship and identify its outline, that is, the problem of initializing a control scheme.

Finally, the work remains of adapting, implementing and evaluating these control schemes in a realistic physical setup, with actual (scale) tugboats that do not affix to the hull of a ship, but is held in place by friction and a control algorithm.

## 5 Physical test setup

### 5.1 General requirements

The physical test setup must be able to perform each of the control schemes described in 4.5, including the alterations described in 4.6. This implies that I need at least six tugboats able to move freely in the plane, at least one ship able to exert thrust from a rear thruster, and a body of water on which to work. The bare minimum of information I need is the absolute position and orientation of the ship, the relative position and orientation of each tugboat, and a measurement of the force applied by each tugboat, as this is assumed known in the described control schemes. I need a way of transferring information to and between tugboats and ship, and both on-board and external computation capabilities, to handle centralized and localized control schemes.

In addition, I want to have the possibility to implement new control schemes, which might require other sensors or actuators to be added to the tugboats, suggesting the need for an adjustable setup entirely within my control. In other words, I want to design my drones from scratch, using reprogrammable microcontrollers. Finally, I want the design to be as simple as possible, to minimize building time.

### 5.2 Design

#### 5.2.1 Drones

I decide to build 10 tugboats. That way I have more than 6 even if some break or run out of power during testing, and I can test out control schemes with significantly more than 6 tugboats. Almost anything floating of appropriate size can serve as the hull of the ship, and I can save time by designing the ship as an outboard motor that attaches to different objects, using the same electronic design as for the tugboats.

To achieve free movement in the plane, I have two good options. I can either make a thrust system that can exert force in any direction, as described by [19], using an azimuth thruster or an omnidirectional water jet, or I can place two fixed thrusters in parallel, using thrust difference to turn. I invoke the golden rule of engineering design, K.I.S.S. (keep it simple, stupid), and go for the second option. While slightly more difficult to control, it is much simpler, and cheaper, to build. A defined front makes force measurements much simpler, and

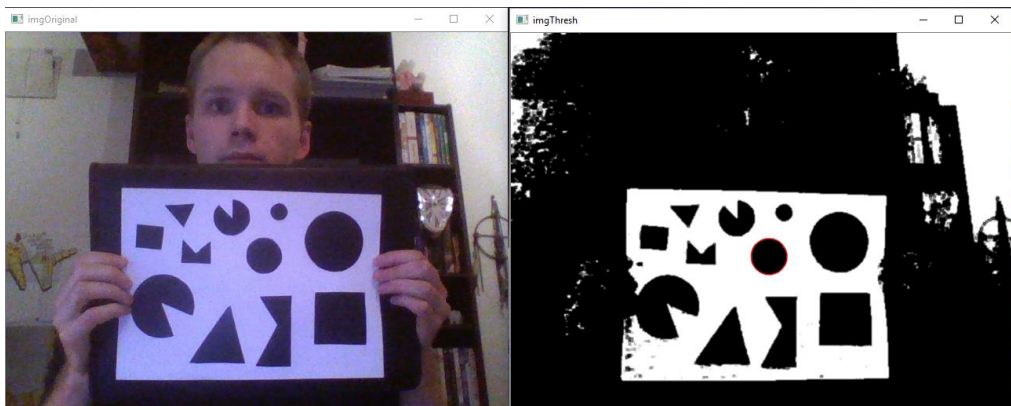
it provides explicit information about thrust direction through orientation measurements. By staying in fresh water, I can mount DC motors in open water directly underneath the hull. This requires only four fixed wires to go through the interface between wet and dry, which can be sealed using hot glue. I design the body of the tugboat round to make certain it is never restricted from rotating.

Real tugboats are lined with tires or pads of soft rubber, which creates a high-friction contact between ship and tug, over large surface area. This effect does not scale that well to very small forces, but a similar effect can be achieved by lining the ship with rough sand paper, and the tugboats with soft rubber like a bicycle hose.

In order to facilitate modifications to the setup at a later stage, I design the circuit board to be modular. The module connected to the motors and power is fixed to the hull, while the control board is detachable. This can be gleamed in figure 12.

### **5.2.2 Instrumentation**

On full-scale tugboats, position and heading is determined using GPS and compass, respectively. GPS is not scalable to robots this size. Instead, I use computer vision. In the wonderful new world of open source software, advanced computer vision is available and usable by mechanical engineering students with limited programming experience. I use a software package called OpenCV, and find two example programs (see Appendix B) that, when combined, lets me uniquely recognize and place more than 22 different shapes using a webcam, with good error margins. If I place two such shapes on each drone, I can calculate the absolute position and orientation of more than 11 separate drones.



**Figure 11:** Notice how the medium-sized circle on the right-hand side image is lined with red. It is recognized as a unique “blob”, and placed in the image.

I also want to measure force exerted by each tugboat, both in order to know whether they are in contact with the ship, and to provide a state-feedback to the tugboat controller, which is responsible for providing the required thrust to the ship. With the design described in 5.2.1, it is sufficient to measure the force on the bow of each tugboat. I want to use a load cell, like that used in a kitchen scale, for this, which provide accurate measurements without the need for calibration.

### 5.2.3 Information flow- and handling

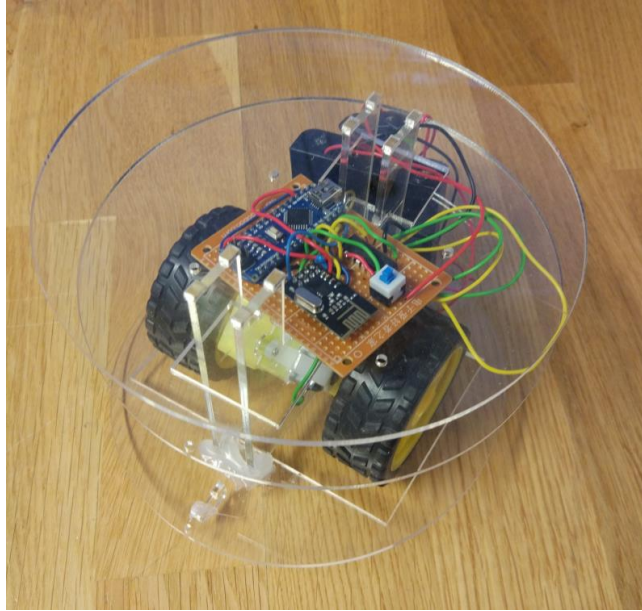
The different control schemes have different communication requirements. I want to cover all bases, so I decide on an all-to-all communication setup. I equip each tugboat, the ship and the computer running computer vision, with a nRF24L01 radio transceiver and an Arduino Nano. The Arduino can interface with sensors, actuators and the transceiver, as well as perform simple on-site calculations.

## 5.3 Progress at project finish

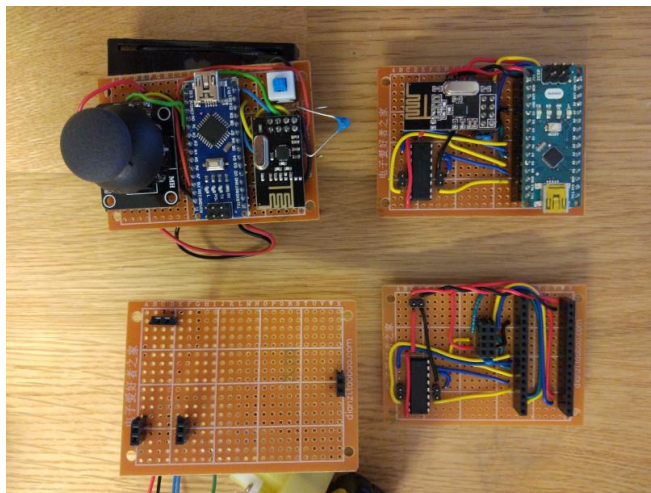
At project finish, I have purchased most of the components required to build the test setup, including a circular swimming pool with a diameter of 3.5 meter. This is sufficiently large to facilitate a roughly 1:200 scale setup. I have validated each component and line of communication separately, except for the load cell and anything involving water. This way I should be able to build the test setup quickly come spring.

I have made the first robot with wheels, see figure 11, because I want to use it to develop what remains on the software side of the setup, before I set up the pool. I have also made a

remote-control to test out driving capabilities, and different programs to test each component during tugboat production.



**Figure 11:** One wheeled tug robot



**Figure 12:** Top left: Remote control. Top right: Control board. Bottom right: Control board without microcontroller or radio transceiver mounted. Bottom left: Lower electronics module, to be fixed to body.

## 6 Preliminary research problems for master thesis

Control theory, specifically adaptive control, seems to be the most promising implementation for tugboat control, given the strict requirements for control when ferrying ships of up to half a million tonnes. A thorough study of different control schemes, focusing on reliability and efficiency, is necessary to be able to make some recommendations to the industry. I will develop my set-based prototype into a development platform for more advanced control algorithms, to save time in the long run, but the real benchmarking will be on the physical setup.

I also want to make a side-study of swarm-based implementations if possible, but I will not make this a priority.

I propose the following research problems for my master's thesis:

1. Build a physical test setup for tugboat control, as described in chapter 6.
2. Implement different control schemes in the test setup, based on the literature discussed in 4.5, and evaluate them in terms of reliability and efficiency.
3. Develop a new control scheme with dynamic tugboat positioning, implemented as a state machine or otherwise, using the ship's main thruster when appropriate.
4. Make recommendations on size, number and control of tugboats for large-scale implementation.

## References

- [1] Beegle, R. "An Overview of Trends in the Tug Market," Marcon International, Inc. (2007)"
- [2] Barker, P. "Unmanned tugs – the ultimate crew protection?," *Maritime Journal*, (2015, Oct. 30). [Online]. Available: <http://www.maritimejournal.com/news101/tugs,-towing-and-salvage/unmanned-tugs-the-ultimate-crew-protection>. [Accessed Dec. 16, 2016].
- [3] Ardito, S., D. Lazarevs, B. Vasiliniuc, Z. Vukic, K. Masabayashi, and M. Caccia. "Cooperative Autonomous Robotic Towing system: definition of requirements and operating scenarios." *IFAC Proceedings Volumes* 45, no. 27 (2012): 262-267.
- [4] Andersen, I. "Nå kommer de første førerløse bussene til Norge," *Teknisk ukeblad*, (2016, Aug. 3). [Online]. Available: <http://www.tu.no/artikler/na-kommer-de-forste-forerlose-bussene-til-norge/349842>. [Accessed Dec. 16, 2016].
- [5] Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., Nagpal, R. "Kilobot: A low cost robot with scalable operations designed for collective behaviors." *Robotics and Autonomous Systems* 62, no. 7 (2014): 966-975.
- [6] Gerstenberg, A., Sjöman, H., Reime, T., Abrahamsson, P., Steinert, M. "A Simultaneous, Multidisciplinary Development and Design Journey–Reflections on Prototyping." In *International Conference on Entertainment Computing*, pp. 409-416. Springer International Publishing, (2015).
- [7] Houde, S., Hill, C. "What do prototypes prototype." *Handbook of human-computer interaction 2* (1997): 367-381.
- [8] Elverum, C. "Advanced Product Development: Prototypes and prototyping." Norwegian University of Science and Technology, (2016).
- [9] Nocedal, J., Wright, S. *Numerical optimization*. Springer Science & Business Media, (2006): 378-380.
- [10] Web page. URL: <https://youtu.be/qv6UVOQ0F44>
- [11] Web page. URL: [https://youtu.be/S9Y\\_I9vY8Qw](https://youtu.be/S9Y_I9vY8Qw)
- [12] Burke, K. "Hacking "Roller Coaster Tycoon" with genetic algorithms and Go." [Conference presentation]. OScon, Portland, OR (Jul. 22, 2015).
- [13] Web page- URL: <http://luyiy.tumblr.com/post/60185103639/guppies-evolving-neural-networks-wip>
- [14] Fossen, T. I. *Marine control systems: guidance, navigation and control of ships, rigs and underwater vehicles*. Marine Cybernetics, (2002).
- [15] Landau, I. D., Lozano, R., M'Saad, M., Karimi, A. *Adaptive control: algorithms, analysis and applications*. Springer Science & Business Media, (2011): 2-19.

- [16] Braganza, D., Feemster, M., Dawson, D. M. *Positioning of large surface vessels using multiple tugboats*. No. CU/CRB/9/24/06/# 1. CLEMSON UNIV SC DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, (2006).
- [17] Van Bui, P., and Kim, Y.B. "Development of constrained control allocation for ship berthing by using autonomous tugboats." *International Journal of Control, Automation and Systems* 9, no. 6 (2011): 1203-1208.
- [18] Bidikli, B., Tatlicioglu, E., Zergeroglu, E. "Robust dynamic positioning of surface vessels via multiple unidirectional tugboats." *Ocean Engineering* 113 (2016): 237-245.
- [19] Esposito, J, Feemster, M., Smith, E. "Cooperative manipulation on the water using a swarm of autonomous tugboats." In *IEEE International Conference on Robotics and Automation*, (2008): 1501-1506.
- [20] Esposito, J. M. "Distributed grasp synthesis for swarm manipulation with applications to autonomous tugboats." In *IEEE International Conference on Robotics and Automation*, (2008): 1489-1494.



## 7 Appendices

### *A. Code for set-based prototype in Processing IDE*

---

```

/**
 * Circular Boats
 * Based on:
 * Bouncy Bubbles
 * based on code from Keith Peters.
 *
 * Multiple-object collision.
 */

int controlMode = 5;
int numTugs = 5;
int tugSize = 30;
int numBoats = numTugs + 1;
//All constants on the order of magnitude 0-100
float spring = 10;
float current = 10;
float friction = 100;
float pi = 3.1415;
Ball[] balls = new Ball[numBoats];
Ball ship = balls[0];
Control control = new Control(controlMode, numTugs, balls);

void setup() {
  size(1280, 720);
  balls[0] = new Ball(random(width), random(height), 100, 0, balls);
  for (int i = 1; i < numBoats; i++) {
    balls[i] = new Ball(random(width), random(height), tugSize, i, balls);
  }
  control.controlSetup();
  stroke(0);
  fill(255, 204);
}

void draw() {
  background(0);
  control.Compute();
  for (Ball ball : balls) {
    ball.collide();
    ball.move();
    ball.display();
  }
}

class Ball {

  float x, y;

  float Fx = 0, Fy = 0;

  float diameter;

  float mass;

```

```

float vx = 0;
float vy = 0;
int id;
PID xPID;
PID yPID;
PID FPID;
Ball[] others;

Ball(float xin, float yin, float din, int idin, Ball[] oin) {
    x = xin;
    y = yin;
    diameter = din;
    mass = diameter*diameter;
    id = idin;
    others = oin;
}

void collide() {
    for (int i = id + 1; i < numBoats; i++) {
        float dx = others[i].x - x;
        float dy = others[i].y - y;
        float distance = sqrt(dx*dx + dy*dy);
        float minDist = others[i].diameter/2 + diameter/2;
        if (distance < minDist) {
            float angle = atan2(dy, dx);
            float overlap = minDist-distance;
            float FxCollide = cos(angle) * overlap * overlap * spring;
            float FyCollide = sin(angle) * overlap * overlap * spring;
            vx -= FxCollide/mass;
            vy -= FyCollide/mass;
            others[i].vx += FxCollide/others[i].mass;
            others[i].vy += FyCollide/others[i].mass;
            if(id == 0){
                Fx += FxCollide;
                Fy += FyCollide;
            }
        }
    }
}

```

```

if(id == 0){
    strokeWeight(4);
    stroke(255, 0, 0);
    line(x, y, x - diameter/200 * Fx, y - diameter/200 * Fy);
    Fx = 0;
    Fy = 0;
}
}

void move() {
    float FxWater = -vx * abs(vx) * friction * diameter / 1000;
    float FyWater = -(vy - current/10) * abs(vy - current/10) * friction * diameter / 1000;
    vx += (FxWater + Fx)/mass;
    vy += (FyWater + Fy)/mass;
    x += vx;
    y += vy;
    if (x + diameter/2 > width) {
        x = width - diameter/2;
        vx *= -1;
    }
    else if (x - diameter/2 < 0) {
        x = diameter/2;
        vx *= -1;
    }
    if (y + diameter/2 > height) {
        y = height - diameter/2;
        vy *= -1;
    }
    else if (y - diameter/2 < 0) {
        y = diameter/2;
        vy *= -1;
    }
}

void display() {
    stroke(0);
    strokeWeight(1);
    ellipse(x, y, diameter, diameter);
}

```

```

strokeWeight(4);
stroke(255, 0, 0);
line(x, y, x + diameter/500 * Fx, y + diameter/500 * Fy);
}
}

```

---

```

class Control{
float Fx = 0;
float Fy = 0;
float F;
float angle;
float arc;
int controlMode;
int numberOfTugs;

Control(int controlModeIn, int numberOfTugsIn, Ball[] ball){
controlMode = controlModeIn;
numberOfTugs = numberOfTugsIn;
}

void controlSetup(){
switch(controlMode) {
case 0: //Complacent - Do nothing
break;
case 1: //Group hug! - converge on ship
break;
case 2: //Out of the way! - converge on ship, but move out of the way if in the ship's path
break;
case 3: //Push from behind - Move behind the ship and push proportionally to the force vector
break;
case 4: //Disperse + DP
arc = 2*pi / (numberOfTugs);
break;
case 5: //Disperse + DP, PID
arc = 2*pi / (numberOfTugs);
for(int i = 1; i<numberOfTugs + 1; i++){
balls[i].xPID = new PID(20.0, 0.0, 2.0, false);
balls[i].yPID = new PID(20.0, 0.0, 2.0, false);
}
break;
}
}

```

```

case 6: //Disperse + DP, PID
    arc = 2*pi / (numberOfTugs);
    for(int i = 1; i<numberOfTugs + 1; i++){
        balls[i].xPID = new PID(10.0, 0.0, 2.0, false);
        balls[i].yPID = new PID(10.0, 0.0, 2.0, false);
    }
    break;
default:
    break;
}
}

void Compute(){
if(controlMode < 100){//Simple algorithms, all tugs are given the same rules
for (Ball ball : balls) {
    ////////////////////////////////////// SHIP //////////////////////////////////////
    if(ball.id == 0){
        if(mousePressed){
            line(mouseX, mouseY, ball.x, ball.y);
            Fx = mouseX - ball.x;
            Fy = mouseY - ball.y;
            F = sqrt(sq(Fx) + sq(Fy));
            angle = atan2(Fy, Fx);
        }
        else{
            Fx = 0;
            Fy = 0;
            F = 0;
        }
        switch(controlMode) {
            case 0: //Complacent - Do nothing
                break;
            case 1: //Group hug! - converge on ship
                break;
            case 2: //Out of the way! - converge on ship, but move out of the way if in the ship's path
                angle = atan2(Fy, Fx);
                arc = min(F/500, 3*pi/4);

```

```

        line(ball.x, ball.y, (ball.x + ball.diameter/2 * cos(angle + arc)), (ball.y + ball.diameter/2 * sin(angle +
arc)));
        line(ball.x, ball.y, (ball.x + ball.diameter/2 * cos(angle - arc)), (ball.y + ball.diameter/2 * sin(angle -
arc)));
        break;
    case 3: //Push from behind - Move behind the ship and push proportionally to the force vector
        arc = min(F/500, 3*pi/4);
        break;
    case 4: //Disperse + DP
        break;
    case 5: //Disperse + DP, PID
        break;
    case 6: //Disperse + DP, PID
        break;
    default:
        break;
}
}
//////////////////////////////////// TUGS //////////////////////////////////////
else{
    float xPos;
    float yPos;
    float dx = balls[0].x - ball.x;
    float dy = balls[0].y - ball.y;
    float dAngle = (atan2(-dy, -dx) - angle); //Angle between desired ship heading and a line between tug
and ship +- 180
    if(dAngle > pi){dAngle -= 2*pi;}
    else if(dAngle < -pi){dAngle += 2*pi;} //dAngle in +- pi
    //println(dAngle * 180/pi);
    //line(balls[0].x, balls[0].y, (balls[0].x + diameter/2 * cos(angle + dAngle)), (balls[0].y + diameter/2 *
sin(angle + dAngle)));

    switch(controlMode) {
        case 0: //Complacent - Do nothing
            ball.Fx=0;
            ball.Fy=0;
            break;
        case 1: //Group hug! - converge on ship
            ball.Fx = dx;

```

```
ball.Fy = dy;
```

```
break;
```

case 2: //Out of the way! - converge on ship, but move out of the way if in the ship's path

```
if(dAngle < arc && dAngle > 0){
```

```
    ball.Fx = 100 * cos(angle + dAngle + pi/2);
```

```
    ball.Fy = 100 * sin(angle + dAngle + pi/2);
```

```
}
```

```
else if(dAngle > -arc && dAngle < 0){
```

```
    ball.Fx = -100 * cos(angle + dAngle + pi/2);
```

```
    ball.Fy = -100 * sin(angle + dAngle + pi/2);
```

```
}
```

```
else{
```

```
    ball.Fx=dx;
```

```
    ball.Fy=dy;
```

```
}
```

```
break;
```

case 3: //Out of the way! Scale and shimmy - Like Out of the way! but tugs will try to be at the very back of the ship, and scale their thrust according to desired ship force

```
//line(balls[0].x, balls[0].y, (balls[0].x + diameter/2 * cos(angle + dAngle)), (balls[0].y + diameter/2 * sin(angle + dAngle)));
```

```
if(!mousePressed){
```

```
    ball.Fx = 5*dx;
```

```
    ball.Fy = 5*dy;
```

```
}
```

```
else if(dAngle > 0){
```

```
    ball.Fx= F/100*dx + 500 * cos(angle + dAngle + pi/2);
```

```
    ball.Fy= F/100*dy + 500 * sin(angle + dAngle + pi/2);
```

```
}
```

```
else{
```

```
    ball.Fx= F/100*dx - 500 * cos(angle + dAngle + pi/2);
```

```
    ball.Fy= F/100*dy - 500 * sin(angle + dAngle + pi/2);
```

```
}
```

```
break;
```

case 4:

```
xPos = balls[0].x + (balls[0].diameter + ball.diameter)/2 * cos(ball.id * arc);
```

```
yPos = balls[0].y + (balls[0].diameter + ball.diameter)/2 * sin(ball.id * arc);
```

```
ball.Fx = (xPos-ball.x)*2 + Fx/50;
```

```
ball.Fy = (yPos-ball.y)*2 + Fy/50;
```

```
break;
```





```

float sampleTime = 50; //ms
float lastTime = 0;
float lastInput = 0;
float iTerm = 0;

float lastOutput;

PID(float KpIn, float KiIn, float KdIn, boolean directionIn){
    Kp = KpIn;
    Ki = KiIn;
    Kd = KdIn;
    reversedDirection = directionIn;
}

float compute(float input, float setpoint){
    float now = millis();
    float dt = now - lastTime;
    if(dt > sampleTime){
        float error = setpoint - input;
        float pTerm = Kp * error;
        iTerm += Ki * error * dt / 1000;
        float dTerm = Kd * (lastInput - input)/(dt/1000);

        lastTime = now;
        lastInput = input;
        if(!reversedDirection){
            output = pTerm + iTerm + dTerm;
            lastOutput = output;
            return(output);
        }
        else{
            output = - pTerm - iTerm - dTerm;
            lastOutput = output;
            return(output);
        }
    }
    else{
        return lastOutput;
    }
}

```

}  
}

## *B. Example code used as basis for computer vision*

---

```

/**
 * OpenCV SimpleBlobDetector Example
 *
 * Copyright 2015 by Satya Mallick <spmallick@gmail.com>
 *
 */

#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    // Read image
    Mat im = imread( "blob.jpg", IMREAD_GRAYSCALE );

    // Setup SimpleBlobDetector parameters.
    SimpleBlobDetector::Params params;

    // Change thresholds
    params.minThreshold = 10;
    params.maxThreshold = 200;

    // Filter by Area.
    params.filterByArea = true;
    params.minArea = 1500;

    // Filter by Circularity
    params.filterByCircularity = true;
    params.minCircularity = 0.1;

    // Filter by Convexity
    params.filterByConvexity = true;
    params.minConvexity = 0.87;

    // Filter by Inertia
    params.filterByInertia = true;
    params.minInertiaRatio = 0.01;

    // Storage for blobs
    vector<KeyPoint> keypoints;

    #if CV_MAJOR_VERSION < 3 // If you are using OpenCV 2

        // Set up detector with params
        SimpleBlobDetector detector(params);

        // Detect blobs
        detector.detect( im, keypoints);
    #else

```

```

// Set up detector with params
Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);

// Detect blobs
detector->detect( im, keypoints);
#endif

// Draw detected blobs as red circles.
// DrawMatchesFlags::DRAW_RICH_KEYPOINTS flag ensures
// the size of the circle corresponds to the size of blob

Mat im_with_keypoints;
drawKeypoints( im, keypoints, im_with_keypoints, Scalar(0,0,255),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );

// Show blobs
imshow("keypoints", im_with_keypoints );
waitKey(0);
}
// RedBallTracker.cpp

#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

#include<iostream>

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int main() {
    cv::VideoCapture capWebcam(0);          // declare a VideoCapture object and associate to
webcam, 0 => use 1st webcam

    if (capWebcam.isOpened() == false) {           // check if
VideoCapture object was associated to webcam successfully
        std::cout << "error: capWebcam not accessed successfully\n\n"; // if not, print error
message to std out
        return(0);
                                                // and exit program
    }

    cv::Mat matOriginal;          // input image
    cv::Mat matProcessed;         // output image

    std::vector<cv::Vec3f> v3fCircles;           // 3 element vector of floats,
this will be the pass by reference output of HoughCircles()

    char charCheckForEscKey = 0;

    while (charCheckForEscKey != 27 && capWebcam.isOpened()) { // until the Esc key is
pressed or webcam connection is lost
        bool blnFrameReadSuccessfully = capWebcam.read(matOriginal); // get
next frame

        if (!blnFrameReadSuccessfully || matOriginal.empty()) { // if frame not read
successfully

```

```

std::cout << "error: frame not read from webcam\n";           // print
error message to std out
break;
                                                                    // and jump out of while loop
    }

    // smooth the image
    cv::GaussianBlur(matOriginal,                               // function input
                    matProcessed,
// function output
                    cv::Size(5,                               // smoothing window width and height in
5),
                    2);
pixels
                    // sigma value, determines how much the image will be blurred

    // filter on color
    cv::inRange(matProcessed,                                  // function input
                cv::Scalar(0, 0, 175),                        // min filtering value
                cv::Scalar(100, 100, 256),                    // max filtering
                matProcessed);
(if greater than or equal to this) (in BGR format)
value (and if less than this) (in BGR format)
// function output

    // smooth again
    cv::GaussianBlur(matProcessed,                             // function input
                    matProcessed,
// function output
                    cv::Size(5,                               // smoothing window width and height in
5),
                    2);
pixels
                    // sigma value, determines how much the image will be blurred

    cv::dilate(matProcessed, matProcessed, cv::getStructuringElement(cv::MORPH_RECT,
cv::Size(5, 5)));
                    // close image (dilate, then erode)
    cv::erode(matProcessed, matProcessed, cv::getStructuringElement(cv::MORPH_RECT,
cv::Size(5, 5)));
                    // closing "closes" (i.e. fills in) foreground gaps

    // fill circles vector with all circles in processed image
    cv::HoughCircles(matProcessed,                             // input image
                    v3fCircles,
// function output (must be a standard template library vector
                    CV_HOUGH_GRADIENT,
// two-pass algorithm for detecting circles, this is the only choice available
                    2,
                    // size of image / this value = "accumulator resolution", i.e. accum res = size
of image / 2
                    matProcessed.rows / 4,                     // min
                    distance in pixels between the centers of the detected circles
                    100,
                    // high threshold of Canny edge detector (called by
cvHoughCircles)

```

```

        50,
        // low threshold of Canny edge detector (set at 1/2 previous value)
        10,
        // min circle radius (any circles with smaller radius will not be returned)
        400);
// max circle radius (any circles with larger radius will not be returned)

for (int i = 0; i < v3fCircles.size(); i++) { // for each circle . . .

                                                    // show ball position
x, y, and radius to command line
        std::cout << "ball position x = " <<
v3fCircles[i][0] // x position of center point of circle
        << ", y = " <<
v3fCircles[i][1] // y
        position of center point of circle
        << ", radius = " << v3fCircles[i][2] <<
"\n"; // radius of circle

                                                    // draw small green circle at
center of detected object
        cv::circle(matOriginal,
                                                    // draw on original image
        cv::Point((int)v3fCircles[i][0],
(int)v3fCircles[i][1]), // center point of circle
        3,
        // radius of circle in pixels
        cv::Scalar(0, 255,
0),
        // draw pure green (remember, its BGR, not RGB)
        CV_FILLED);

        // thickness, fill in the circle

                                                    // draw red circle around the
detected object
        cv::circle(matOriginal,
                                                    // draw on original image
        cv::Point((int)v3fCircles[i][0],
(int)v3fCircles[i][1]), // center point of circle
        (int)v3fCircles[i][2],
        // radius of circle in pixels
        cv::Scalar(0, 0,
255),
        // draw pure red (remember, its BGR, not RGB)
        3);

        // thickness of circle in pixels
    } // end for

        // declare windows
        cv::namedWindow("Original", CV_WINDOW_AUTOSIZE); // note: you can use
CV_WINDOW_NORMAL which allows resizing the window

```

```
        cv::namedWindow("Processed", CV_WINDOW_AUTOSIZE);           // or
CV_WINDOW_AUTOSIZE for a fixed size window matching the resolution of the image


                                                                    // CV_WINDOW_AUTOSIZE is the default

        cv::imshow("Original", matOriginal);                       // show windows
        cv::imshow("Processed", matProcessed);

        charCheckForEscKey = cv::waitKey(1);                       // delay (in ms) and
get key press, if any
    } // end while

    return(0);
}
```

## C. Risk assessment

NTNU HMS	Kartlegging av risikofylt aktivitet	Utarbeidet av HMS-avd. Godkjent av Rektor	Nummer HMSRV2601	Dato 22.03.2011 Erstatter 01.12.2008	
-------------	-------------------------------------	--	---------------------	---	---


Enhet: IPM Dato: 06.09.2016

Linjeleder: Torgeir Welø


Deltakere ved kartleggingen (m/ funksjon): Sondre Næss Midtskogen (student), Martin Steinert (veileder)  
*(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse)*

Kort beskrivelse av hovedaktivitet/hovedprosess: Prosjekt/Masteroppgave Sondre Næss Midtskogen. Autonomus tugboat operations.

Er oppgaven rent teoretisk? (JA/NEI): NEI *«JA» betyr at veileder innestår for at oppgaven ikke inneholder noen aktiviteter som krever risikovurdering. Dersom «JA». Beskriv kort aktivitetene i kartleggingskjemaet under. Risikovurdering trenger ikke å fylles ut.*

Signaturer: Ansv. veileder:  Student: *Sondre Midtskogen*

ID nr.	Aktivitet/prosess	Ansv. veileder	Eksisterende dokumentasjon	Eksisterende sikringstiltak	Lov, forskrift o.l.	Kommentar
1	Fresing		Godkjent kurs	Bruk av sikkerhetsutstyr		
2	Dreiling		Godkjent kurs	Bruk av sikkerhetsutstyr		
3	Sveising		Godkjent kurs	Bruk av sikkerhetsutstyr		
4	Lodding			Bruk av avtrekk		
5	Bruk av håndhåldte verktøy					
6	Bruk av andre maskiner			Bruk av relevant sikkerhetsutstyr		


NTNU HMS	Risikovurdering	Utarbeidet av HMS-avd. Godkjent av Rektor	Nummer HMSRV2601	Dato 22.03.2011 Erstatter 01.12.2008	
-------------	-----------------	--	---------------------	---	---

Enhet: IPM Dato: 06.09.2016

Linjeleder: Torgeir Welø

Deltakere ved kartleggingen (m/ funksjon): Sondre Næss Midtskogen (student), Martin Steinert (veileder)  
*(Ansv. Veileder, student, evt. medveiledere, evt. andre m. kompetanse)*

Risikovurderingen gjelder hovedaktivitet: Prosjekt/Masteroppgave Sondre Næss Midtskogen. Autonomus tuboat operations.

Signaturer: Ansv. veileder:  Student: *Sondre Midtskogen*

ID nr	Aktivitet fra kartleggings-skjemaet	Mulig uønsket hendelse/ belastning	Vurdering av sannsynlighet (1-5)	Vurdering av konsekvens:				Risiko-Verdi (menn-eske)	Kommentarer/status Forslag til tiltak
				Menneske (A-E)	Ytre miljø (A-E)	Øk/ materiell (A-E)	Om-gjems (A-E)		
1	Fresing	Kutt- og klemskader	1	A	A	A	B	A1	Bruk verneutstyr og generell forsiktighet (for alle aktiviteter)
2	Dreiling	Klær, hår, kroppsdeler hektes i dreiestykket, og blir dratt inn i maskinen	1	C	A	A	C	C1	Hold alle lemmer og andre løsthengende objekter vekk fra maskinen til enhver tid ALLTID bruk sveisemaske, hansker og overall
3	Sveising	Brannskader, øyeskader	2	D	A	A	D	D2	
4	Lodding	Brannskader, skader fra inhalering av avgasser	2	A	A	A	A	A2	Bruk øvsug, sett loddejem på plass når det ikke brukes aktivt
5	Bruk av håndhåldte verktøy	Kutt- og klemskader	3	A	A	A	A	A3	Bruk vernebriller, arbeidshansker, vernesko hvis fornuftig
6	Bruk av andre maskiner	Kutt, klem- og brannskader	3	A	A	A	A	A3	Bruk vernebriller, arbeidshansker, vernesko hvis fornuftig



NTNU	Risikovurdering	Utbildet av	Nummer	Dato	
		HMS-ansv.	HMSRV2601	22.03.2011	
HMS		Godkjent av		Erstatler	
		Rektor		01.12.2006	

Sannsynlighet vurderes etter følgende kriterier:

Svært liten 1	Liten 2	Middels 3	Stor 4	Svært stor 5
1 gang pr 50 år eller sjeldnere	1 gang pr 10 år eller sjeldnere	1 gang pr år eller sjeldnere	1 gang pr måned eller sjeldnere	Sjår ukontlig

Konsekvens vurderes etter følgende kriterier:

Gradering	Menneske	Ytre miljø Vann, jord og luft	Økologi	Omdømme
<b>E</b> Svært alvorlig	Død	Svært langvarig og ikke reversibel skade	Drifts- eller aktivitetsstans >1 år	Troverdighet og respekt betydelig og varig svekket
<b>D</b> Alvorlig	Alvorlig personskade. Mulig uferhet.	Langvarig skade. Lang restitusjonstid	Driftstans > ½ år Aktivitetstans i opp til 1 år	Troverdighet og respekt betydelig svekket
<b>C</b> Moderat	Alvorlig personskade.	Mindre skade og lang restitusjonstid	Drifts- eller aktivitetsstans < 1 mnd	Troverdighet og respekt svekket
<b>B</b> Liten	Skade som krever medisinsk behandling	Mindre skade og kort restitusjonstid	Drifts- eller aktivitetsstans < 1 uke	Negativ påvirkning på troverdighet og respekt
<b>A</b> Svært liten	Skade som krever førstehjelp	Ubetydelig skade og kort restitusjonstid	Drifts- eller aktivitetsstans < 1 dag	Liten påvirkning på troverdighet og respekt

Risikoverdi = Sannsynlighet x Konsekvens

Beregn risikoverdi for Menneske. Enheten vurderer selv om de i tillegg vil beregne risikoverdi for Ytre miljø, Økologi/materiell og Omdømme. I så fall beregnes disse hver for seg.

Til kolonnen "Kommentarer/status, forslag til forebyggende og korrigerende tiltak":

Tiltak kan påvirke både sannsynlighet og konsekvens. Prioriter tiltak som kan forhindre at hendelsen inntreffer, dvs. sannsynlighetsreducerende tiltak foran skjerpet beredskap, dvs. konsekvensreducerende tiltak.

NTNU	Risikomatrix	Utbildt av	Nummer	Dato	
		M-ansv.	HMSRV2604	08.02.2010	
HMS/RS		Godkjent av		Erstatler	
		Rektor		09.02.2010	

MATRISSE FOR RISIKOVURDERINGER ved NTNU

KONSEKVENSEN	Svært alvorlig	E1	E2	E3	E4	E5
	Alvorlig	D1	D2	D3	D4	D5
	Moderat	C1	C2	C3	C4	C5
	Liten	B1	B2	B3	B4	B5
	Svært liten	A1	A2	A3	A4	A5
	Svært liten	Liten	Middels	Stor	Svært stor	
SANNSYNLIGHET						

Prinsipp over akseptkriterium. Forklaring av fargene som er brukt i risikomatriksen.

Farge	Beskrivelse
Rød	Uakseptabel risiko. Tiltak skal gjennomføres for å redusere risikoen.
Grøn	Vurderingsområde. Tiltak skal vurderes.
Gul	Akseptabel risiko. Tiltak kan vurderes på fra andre hensyn.