NTNU

Norwegian University of
Science and Technology

Perception-Driven Obstacle-Aided Locomotion for snake robots, linking virtual to real prototypes

Stian Grøttum Danielsen

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Øyvind Stavdahl, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

Master's Thesis

Student's name:	Stian Grøttum Danielsen
Field:	Engineering Cybernetics
Title (Norwegian):	Persepsjonsdrevet hinderassistert lokomosjon for slangeroboter, kobling fra virtuelle til ekte prototyper
Title (English):	Perception-Driven Obstacle-Aided Locomotion for snake robots, linking virtual to real prototypes

Description:
Snake robots equipped with sensors and tools could potentially contribute to applications such as fire-fighting, industrial inspection, search-and- rescue and more. Such capabilities would require that a snake robot has a high degree of awareness of its surroundings and is able to exploit objects and irregularities in its environment to gain propulsion.

The term perception-driven obstacle-aided locomotion may be adopted to define locomotion where the snake robot utilizes a sensory-perceptual system to exploit the surrounding operational space and identifies walls, obstacles or other external objects, for means of propulsion. SINTEF and NTNU have already produced some initial results within this area and have ongoing projects to further develop snake robot capabilities.

Our research group is developing a simplified snake robot model to deal with a lower-dimensional system that allows for establishing the foundation elements of perception-driven obstacle-aided locomotion. In particular, the obstacle triplet model was recently proposed (with 3 simultaneous push-points at alternating sites of the path). In this model, only one control variable for the torque is needed to achieve obstacle-aided locomotion. This propulsive torque can be applied at any point along the path and it can be seen as a thruster for the snake robot. This method contributes towards the preliminary stage for realizing perception-driven obstacle-aided locomotion for snake robots.

To give researchers the possibility of testing different control algorithms for perception-driven obstacle-aided locomotion in a realistic laboratory setup, a basic simulation framework is under development. This framework is based on the use of the Gazebo 3D simulator and the Robot Operating System (ROS). Gazebo is used to simulate the robot operational scenario, while ROS serves as the interface for the robot.

1. Improve the control method for perception-driven obstacle-aided locomotion
2. Implement a communication interface between the ROS framework and the real snake robot
3. Verify the implementations through simulations and experiments

Advisor(s):	Øyvind Stavdahl, Assoc. Professor, Dept. of Engineering Cybernetics, NTNU Filippo Sanfilippo, Post Doc., Dept. of Engineering Cybernetics, NTNU

Trondheim, <12.12.2016>

Øyvind Stavdahl
Supervisor

Summary

Snake robots have the potential to be used for many tasks where maneuvering through narrow and uneven terrain is necessary. Examples of such tasks include search-and-rescue missions, firefighting and pipe inspection. Real snakes exploit the unevenness in the terrain by strategically pushing on objects to gain forward propulsion. Research on adopting this strategy for snake robots is fairly new, and there are still many problems to be solved. In this thesis a control method for perception-driven obstacle-aided locomotion for snake robots is presented. The control method takes the multidimensional problem of controlling each joint of the robots and reduces it to a lower dimensional problem. Two main contributions were made as part of this work. The first contribution was the improvement of a previous implementation of the control method for a simulated snake robot, mainly by introducing a shape controller to aid the forward motion of the robot. The second contribution was the implementation of a communication interface between the control framework and the interface of a real snake robot.

The control method was implemented in the ROS framework and tested in the robot simulator Gazebo to evaluate the performance of the design. The real snake robot was controlled through an interface in LabVIEW. The communication interface was implemented so that ROS and LabVIEW could exchange control data for both the virtual and the real snake robot. Simulations and experiments were performed to validate the implementations.

Sammendrag

Slangeroboter har potensialet til å bli brukt til mange oppgaver hvor manuvrering gjennom trangt og ujevnt terreng er nødvendig. Eksempler på slike opppgaver er redningstjenester, brannslukning og rørinspeksjon. Ekte slanger utnytter ugjenvheten i terrenget ved strategisk ved å dytte på objekter for å oppnå framoverdrift. Forskningen på å bruke denne strategien for slangeroboterer er ganske ny, og det er fremdeles mange problemer som må løses. In denne oppgaven blir en styringsmetode for persepsjonsdrevet hinderassistert lokomosjon for slangeroboter presentert. Styringsmetoden tar det flerdimensjonale problemet ved styre hvert ledd av en slangerobot og reduserer det til et problem av lavere dimensjon. To hovedbidrag ble gjort som del av denne oppgaven. Det første bidraget var forbedringen av en tidligere implementasjon av en styringsmetode for en simulert slangerobot, hovedsaklig ved å introdusere en form-kontroller for å hjelpe med framovebevegelsen til roboten. Det andre bidraget var implementasjonen av et kommunikasjonsgrensesnitt mellom styringsrammeverket og grensesnittet til en ekte slangerobot.

Styringsmetoden ble implementert i rammeverket ROS og testet i robotsimulatoren Gazebo for å evaluere ytelsen på designet. Den ekte slangeroboten var styrt gjennom et grensesnitt i LabVIEW. Kommunikasjonsgrensesnittet var implementert slik at ROS og LabVIEW kunne utveksle styringsdata for både den virtuelle og den ekte slangeroboten. Simuleringer og eksperimenter ble gjennomført for å validere implementasjonen.

Preface

This work is my master's thesis, and the culmination of my five years studying Cybernetics and Robotics at NTNU. From the preliminary project that started in the fall semester of 2016 up until the completion of this thesis, there have been many challenges and a lot of hard work. I have learned a lot, and I value this experience greatly.

I would like to thank Filippo Sanfilippo, the co-supervisor for my master's thesis, for all his help and guidance. He showed genuine interest in my work, and motivated me to do my best.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Snake robots

This section is copied from the author's previous work (Danielsen, 2016).

Researchers have for a long time been looking to nature for discovering new ways to solve challenging tasks. In robot technology, there are many examples of robots that have been developed based on animals. In the animal kingdom there are a variety of movement methods to be found, many of which are more versatile than driving on wheels. For navigating terrain for example, wheel- or belt-driven robots can easily get stuck or unable to turn. While four-legged robots like the robot cheetah in (Wang et al., 2012) are more able to move in uneven terrain, a movement method even more adaptable to the environment can be found in some limbless animals. Snakes have the ability to move in tight spaces, shaping themselves to fit between obstacles and around them. Not only can they move between obstacles, they utilize their environment by pushing on rocks, branches and other uneven surfaces in the terrain. They can also swim, climb trees, and some even fly (Perrow and Davy, 2008). For these reasons, snake robots are being researched to solve many practical problems like inspecting pipes and traversing rubble in search and rescue operations.

There have already been designed control systems allowing snake robots to move forwards on a flat ground. In many of the considered applications for snake robots, like search and rescue, the ability to move between obstacles is essential. Methods have been developed to allow the snake robots to avoid obstacles, but they lack efficiency because they do not utilize the obstacles for propulsion. The fastest snakes in nature are known to exploit the terrain for locomotion (Transeth et al., 2009). If snake robots can be made to move at reasonable speeds through terrain and tight spaces, they can be used effectively for fire fighting and search and rescue missions. In order to utilize the obstacles, many systems need to be in place. First of all, the obstacles must be identified in order to make any navigation decisions. Secondly, a planned path for traversing the terrain needs to be generated. What then remains is a control system to make sure the snake robot stays on its path while also pushing forward on the obstacles to gain propulsion. All of this together is called

perception-driven obstacle-aided locomotion, as it perceives the surrounding environment by use of sensors, and then uses the obstacles to gain locomotion.

1.2 A control approach

Snake robots have high degrees of freedom due to their many joints, making control a challenging task. A recently developed approach to perception-driven obstacle-aided locomotion in the case of three obstacles on alternating sides of the snake robot is described in (Sanfilippo et al., 2016b), where the control problem is reduced to lower dimensional problem. This approach is the topic for this thesis as it has not yet been implemented in a fully working system. The purpose of this thesis is to develop a prototype of the control method described in the approach. This control prototype is later to be integrated with a system for obstacle-perception.

1.3 Previous work

This master thesis is a continuation of the work done in the author's and Guillaume Adam's specialization projects during the fall semester of 2016. In that project, a control system for perception-driven obstacle-aided locomotion for snake robots was designed and implemented. The implementation was tested and verified in a simulated environment using the robot simulator Gazebo. The control system was designed for a planar snake robot of 11 links, moving between three obstacles. Some propulsion of the snake robot was achieved, and there was room for improvement on several parts. See (Danielsen, 2016) for details on the implemented control method.

1.4 Contributions

The first improvement in this thesis over my previous work was of the propulsion control system, making it run better in the simulated environment. This was done in several steps. The first was to work on the low level implementation and generalize the module for handling collision sensor information, to allow for an arbitrary number of obstacles to be in contact with the robot while still generating propulsion. Second step was to implement shape control in order to follow a desired path and stay on the track between the obstacles. A simple path planner was also implemented in order to be able to verify the functionality of the shape controller. The shape controller was also useful for helping the snake robot stay in touch with the three obstacles needed by the propulsion controller. To allow for easier understanding of the working forces on the snake robot, a previously made visualization module was improved and generalized to work for an arbitrary number of collisions.

The second contribution was to create a communication interface for linking the ROS control framework with the control framework for a real snake robot.

The last contribution was to do simulations and experiments to verify the implementations, and to show the potential of the framework. Hopefully the reader will be convinced that the framework implemented as part of this thesis is a step towards a fully working

control system, and that it will allow further research on perception-driven obstacle-aided locomotion for snake robots to be carried out with convenience.

1.5 Outline of the report

This paper is organized as follows. Tools and methods that were used in this project are described in Chapter 2. Related research is presented in Chapter 3. Chapter 4 presents the top-level control framework for snake robot locomotion that this work contributes to realizing. Chapter 5 presents the theory behind the control method we are using for perception-driven obstacle-aided locomotion. The implementation of the controllers and other necessary modules is described in Chapter 6. Chapters 7 and 8 contain the descriptions of how the simulations and experiments were carried out, as well as the results obtained from these. A discussion of the results is found in Chapter 9, and the conclusion is found in Chapter 10. Appendix contains information about demonstration videos and other attachments to the report.

It is assumed that the reader has at least basic knowledge in control theory. It should also be noted that the word *snake robot* is many times shortened to simply *snake*, and that the words *collision* and *contact* are interchangeably used for describing the state in which the snake robot touches another object.

Chapter 2

Tools and methods

2.1 Prototype and simulation

The prototype was implemented within the Robotic Operating System (ROS) framework due to several reasons. ROS is widely used in the robot research community, and is by many considered a standard for robot development. ROS has a lot of great built-in tools, reducing the need for programming lots of complex functionality from scratch. It is good for modular design, allowing programmers to separate different parts of the total system into so-called nodes which can operate independently. Modular design is a key element for developing this prototype, as other researchers will later add functionality to the system presented in this report.

For testing the prototype, a robot simulator was used rather than a real robot. There are several advantages of using a simulator when prototyping a control system. Firstly, testing can be done a lot more efficiently, saving time which can be used for further development of the prototype. The simulator takes very little time to initialize, as one just has to launch a program on the computer, compared to a physical system where one might have to connect components together and move the potentially heavy hardware to the position of operation. For safety reasons, more than one person usually have to be present during testing, increasing the time spent for each test. A simulator is also less dangerous, with no risk of damage to people or expensive equipment. Aside from breaking the equipment, repeated usage will lead to wear and tear on components that will have to be replaced, making experiments on the real robot more expensive than simulated experiments.

For this project the robotics simulator Gazebo was chosen. There are several other simulators available, like V-REP, Webots, RoboDK etc., but one of the main advantages of Gazebo is that it's easy to integrate with ROS (Nogueira, 2014).

2.2 ROS

This section is copied from (Danielsen, 2016).

ROS is currently only supported on the operating system Ubuntu, and its interface is command line based. ROS lets you create your own plugins which are run as nodes. A node is defined as a unit that does some form of computation, and can be seen as a module in a system. The nodes communicate with each other by publishing or subscribing to topics as in Figure 2.1. The topics act like buses over which the nodes can send and receive messages. The messages are anonymous, and there is no limit to how many nodes can publish or subscribe to each topic. Services are used instead of messages if a node requires a reply to a request it sends to another node. ROS uses a combination of Xacro (XML macro language), Python and C++. It has support for Python and C++ when making plugins, both languages having their own unique qualities. In this project, ROS Indigo with plugins written in C++ was used to allow easier integration between all the subsystems that were designed independently. There are also a lot of ready-made packages for ROS that can be used with little coding. A package that is used in this work is *ros_control*, a package that has built-in functionality for controlling joint motors by PID controllers, which also takes care of communication between ROS and Gazebo.

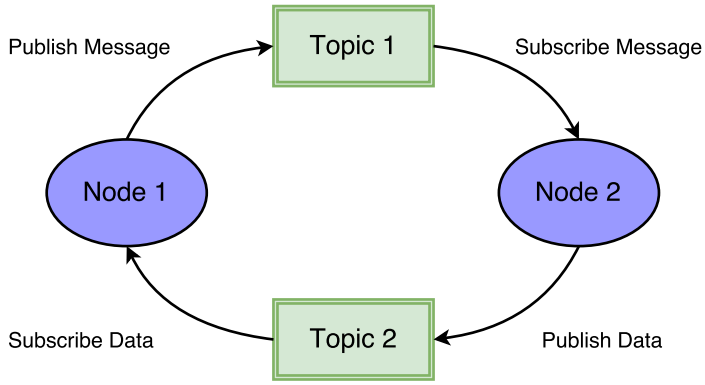


Figure 2.1: Example of how nodes communicate by publishing and subscribing to topics. Node 1 publishes a message to Topic 1, and Node 2 receives it by subscribing to the same Topic.

2.3 Gazebo

This section is copied from (Danielsen, 2016).

Gazebo is a 3D physics simulator used for robot design. It allows easy integration with ROS, letting robot developer test their systems on virtual robots before implementing them on real ones. Figure 2.2 shows the graphical user interface (GUI) of Gazebo, along with a simple model of a snake robot.

Gazebo 2.2 was used in this project, as it is easily integrated with ROS Indigo.

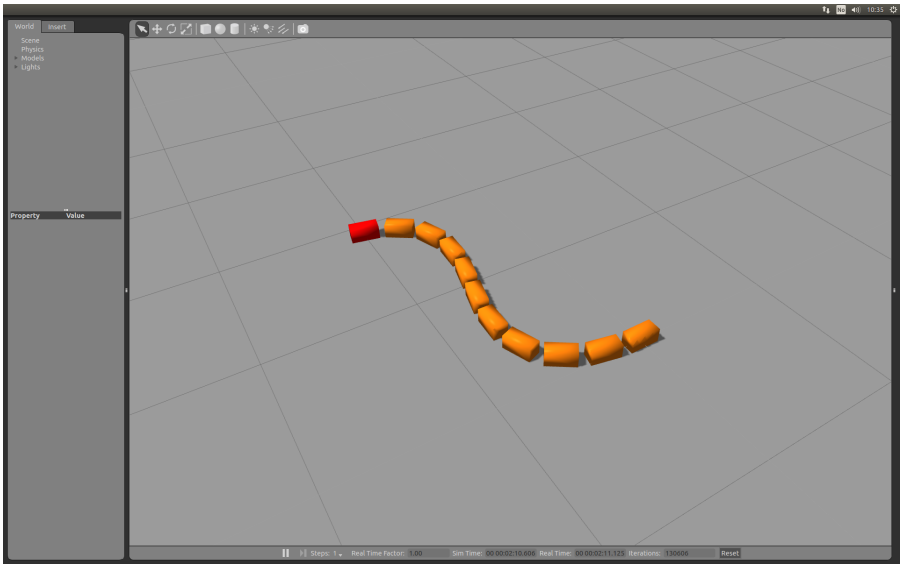


Figure 2.2: The GUI of Gazebo. The robot shown is the snake used in this project.

2.4 LabVIEW

LabVIEW is a system design platform for a graphical programming language. When programming with LabVIEW, the programmer works on the front- and back end of the software simultaneously. The window for creating the back end is called a block diagram, while front end is called a front panel. The front panel is where the user will interact with the system when the finished program is run. LabVIEW supports several kind of dials, buttons, switches, and other ways to input data to your system. The output can be given by lights, meters, gauges, number indicators etc. All of these input and output mechanisms will be represented as input and output data in the block diagram, where you implement your program graphically connecting blocks together. In Figure 2.3 an example of a program implemented with LabVIEW is shown. A front panel and block diagram is together called a VI, and is represented as a block if you use it within another VI.

2.5 ROSforLabVIEW

ROSforLabVIEW is a software package containing LabVIEW VIs that allow you to create ROS nodes, publishers and subscribers. It was developed at Tufts University as a solution for communicating with ROS applications. With this package installed next to LabVIEW, one can communicate with other ROS nodes designed outside of LabVIEW. The nodes, publishers and subscribers come in the form of VIs which one can input to a block diagram.

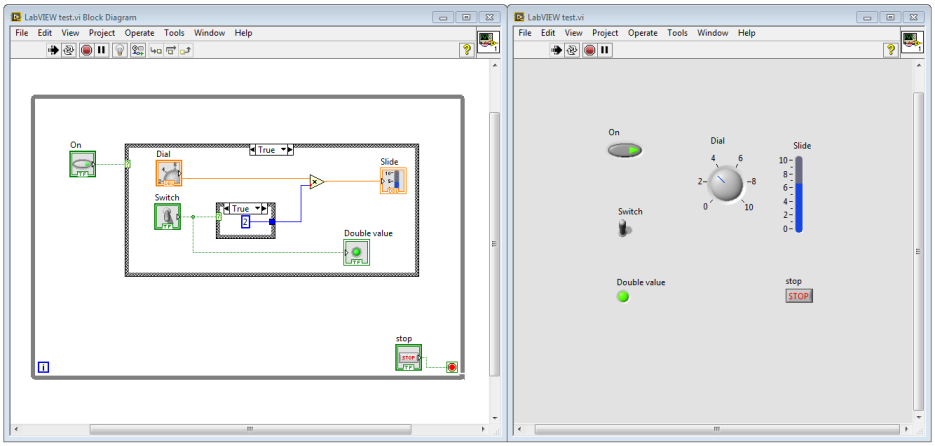


Figure 2.3: Example program implemented in LabVIEW. The left window is the block diagram where all the main programming is done. The right window shows the front panel for the controlled process, where one can both input and read data.

2.5.1 Nodes and topics

A ROS node will be automatically generated with a default name when the first topic has been initialized. Initializing more than one topic will by default not create more nodes, but it can be done by changing the default node name associated with the topics. To initialize a topic, one can use a *ROS_Topic_Init.vi* block. The topic initializing block has one input for a cluster containing information about the topic name, message type, action (publisher or subscriber), update frequency and size of write queue, and another input for a node it can be connected to. The block outputs a cluster with information about the topic and the node it belongs to, and an error message if something went wrong. This information goes into either a *ROS_Topic_Read.vi* or *ROS_Topic_Write.vi* block. These blocks take care of subscribing and publishing to the given topic. Finally, *ROS_Topic_Close.vi* is called, and the subscriber/publisher for this topic is deleted. Examples of a publisher and a subscriber can be seen in Figures 2.4 and 2.5 respectively.

2.5.2 Parsing messages

ROSforLabVIEW supports many of the standard ROS message types, which you can build and parse with the VI's supplied with the package. One can also use custom messages types. In order to parse messages containing several variables of either the same or different data types, parsing blocks for each variable must be placed in series, as each message parsing block extracts one variable before forwarding a message containing the remaining data. In order to parse unsupported message types, such as messages containing other messages, one must create a new parsing function. Parsing the message means unflattening (converting) a string of binary values to a value of the desired data type. Examples on how to do this can be seen when opening the built-in parsing blocks.

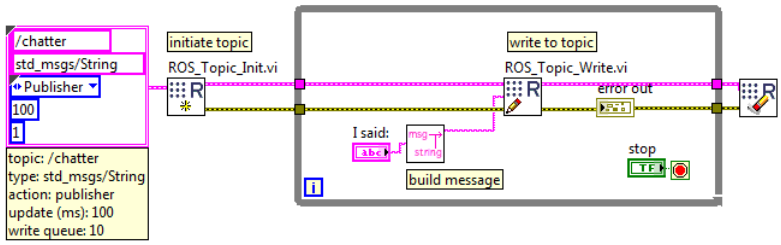


Figure 2.4: An example of the block diagram of a ROS publisher in LabVIEW. A message containing a data value of type *string* is published on the topic */chatter*. This example program comes with the ROSforLabVIEW package.

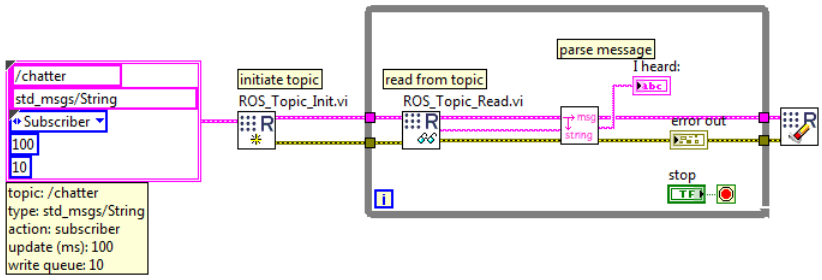


Figure 2.5: An example of the block diagram of a ROS subscriber in LabVIEW. A message containing a data value of type *string* is received from the topic */chatter*. This example program comes with the ROSforLabVIEW package.

2.5.3 Building messages

Building messages is just the opposite of parsing them. For supported message types, one can use the built-in message building blocks. The blocks are to be connected in series just like the parsing blocks. A message building block has an input for a message and data to be appended to the message. The output of the block will be the input message with the new data appended to it. As with parsing, if one wishes to put a message into another message, a new function for this must be made. Building a message means flattening (converting) the data value to a string of binary values before appending the string to the message. Examples of how to build a message can be seen inside the built-in parsing blocks.

2.6 Snake robot

The real (i.e. not simulated) snake robot consists of 13 links connected by 12 single-axis rotating joints. Seven of the joints rotate in pitch, while the remaining six rotate in yaw. It is equipped with passive wheels and a camera on the front. The joint motors are connected to an external controller and power supply through a cable coming out from the tail. The robot is controlled through an interface in LabVIEW.

Chapter 3

Literature Review

3.1 Snake motion

This chapter is copied from (Danielsen, 2016).

To this date, a lot of research has been done on movement of snake robots. A review on previous research leading up to perception-driven obstacle-aided locomotion was done in (Sanfilippo et al., 2016a), and is the basis for the list of research presented here. In the early stages, basic motion pattern similar to those found in nature were being tested. In (Hirose and Mori, 2004), three categories of movement are presented. These include serpentine motion (also called lateral undulation), the most common movement pattern as seen in Figure 3.1. Other movements allowing the snake robot to roll and swim were successfully implemented. This laid the foundation for serpentine robot control.

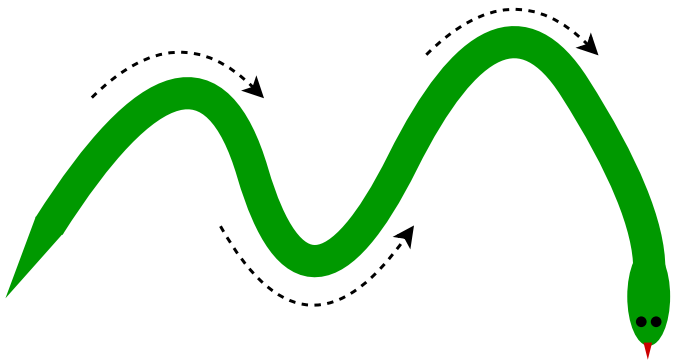


Figure 3.1: The movement pattern of lateral undulation

3.2 Obstacle avoidance

A natural approach to handling obstacles is to avoid them altogether. Hitting obstacles may lead to mechanical stress and damage of equipment. For this reason, a lot of research have focused on obstacle avoidance during locomotion. Artificial Potential Field (APF) theory (Lee and Park, 2003) have been tried. In this approach, imaginary force fields are modeled around objects. These forces are either repulsive or attractive. A motion controller based on APF was presented in (Ye et al., 2010). A problem with the standard APF approach is that the robot may get stuck in a local minima, unable to move. Solutions to this problem were presented in (Yagnik et al., 2010) and (Nor and Ma, 2014).

3.3 Obstacle accomodation

A more relaxed approach to obstacle avoidance can be achieved by using sensory feedback. In (Shan and Koren, 1993), a motion planning system was implemented to allow a snake-like robot to accommodate environmental obstructions. A general formulation of the motion constrains are presented in (Shan and Koren, 1995). This formulation led to a new inverse kinematic model that could be used to find the joint motion for snake robots under constraints.

3.4 Obstacle-aided locomotion

Real snakes on the other hand do not avoid obstacles. They exploit the irregularities in the terrain to gain effective forward locomotion. The snakes bend themselves around the obstacles, and all the sections of the body follow the path of the head (Gray, 1946). Obstacle-aided locomotion (Transeth et al., 2008), (Holden et al., 2014a), is therefore considered a key aspect of practical snake robots. According to (Sanfilippo et al., 2016a), little research has been done concerning the possibility of applying this locomotion approach to snake robots.

In (Bayraktaroglu and Blazevic, 2005), a formal definition of motion patterns and their requirements were presented. It was shown that for lateral undulation, the following conditions are required for movement:

- It occurs over irregular ground with vertical projections
- Propulsive forces are generated from the lateral interaction between the mobile body and the vertical projections of the irregular ground, called push-points.
- At least three simultaneous push-points are necessary for this type of motion to take place.
- During the motion, the mobile body slides along its contacted push-points.

Based on these conditions, the author of the same work tested a simple control law for a generic planar mechanism in dynamics simulations. These findings where then applied in a practical experiment in (Bayraktaroglu et al., 2006) through lateral undulation

on a wheel-less snake-like mechanism. An accurate sensing approach was introduced in (Gupta, 2007), designing a robotic snake capable of force sensing. The snake robot was able to move in the horizontal plane by pushing on the obstacles. In (Andruska and Peterson, 2008), the robot actuators were used as force sensors, allowing the snake robot to move through a elastically deformable channel without the need of external tactile senors.

A control method with predetermined and fixed pushing pattern was presented in (Kamegawa et al., 2012). With this method, a few neighbouring joints in addition to the adjacent joints of the contact points were set to make a pushing effort. The propulsion comes from setting the joint torques asymmetrically. A more general version of the method that prevents the snake robot from getting stuck an obstacle-dense area was proposed by the same research group in (Kamegawa et al., 2014).

It is important the snake robot can keep a desired shape when moving through an environment with obstacles. For body shape control, a framework for general motion planing was presented in (Liljebäck et al., 2014). Mass-spring-damper dynamics was assigned to the shape curve defining the motion of the robot. The framework was tested for straight line path following control and body shape compliance in environments with obstacles to demonstrate how it can be applied.

According to a remark in (Sanfilippo et al., 2016a), ”Most of the previous studies highlight the fact that lateral undulation is highly dependent on the actuator torque output and environmental friction.” Some different approaches were discussed in (Ma et al., 2004) and (Holden et al., 2014b) based on this remark. An interesting result from the approach in (Holden et al., 2014b) is that one can check the quality of a given path. That is, one can check which useful forces could be generated during contact with obstacles in the path.

Chapter 4

Control framework

This chapter in short describes the general framework for controlling the snake robot.

4.1 A hierarchical control framework

To achieve a fully autonomous snake robot, not only controllers are needed. Guidance and navigation systems are also imperative to enable the robot to decide where to move, and how to get there. Figure 4.1 illustrates the information flow for the complete system of an autonomous snake robot. This thesis focuses mainly on the control part of the implementation.

Figure 4.2 shows the hierarchical control framework proposed in (Sanfilippo et al., 2016b) for achieving perception-driven obstacle-aided locomotion in snake robots. The framework is divided into four different abstraction levels:

- Perception is responsible for all the sensing of the environment, and produce a representation of the surroundings. Both foreign objects and the robot itself are given geometrical properties, and given positions relative to the map. The expected output is the relative positions and other properties of obstacles in the terrain.

- The motion planning level takes care of decision making, and path planning. The level uses the input from the perception level in order to make decision about where to go, and which route to use. Human users may also input commands for steering the robot. The output from this level is the snake robot's trajectory.

- High-Level control uses the desired trajectory and information about the obstacles to identify necessary contact forces on the obstacles. The output from this level is control input for the lower level.

- The bottom abstraction is low-level control. That level takes care of the low-level control of each joint.

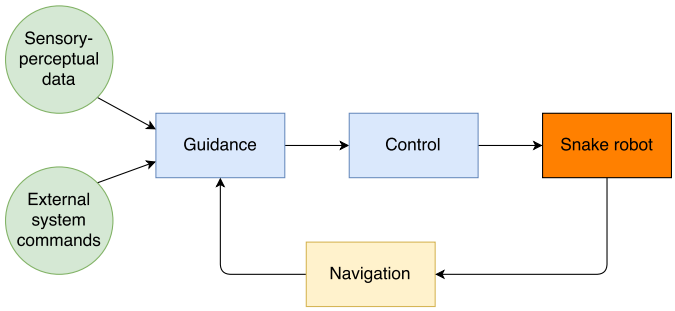


Figure 4.1: Information flow for an autonomous snake robot

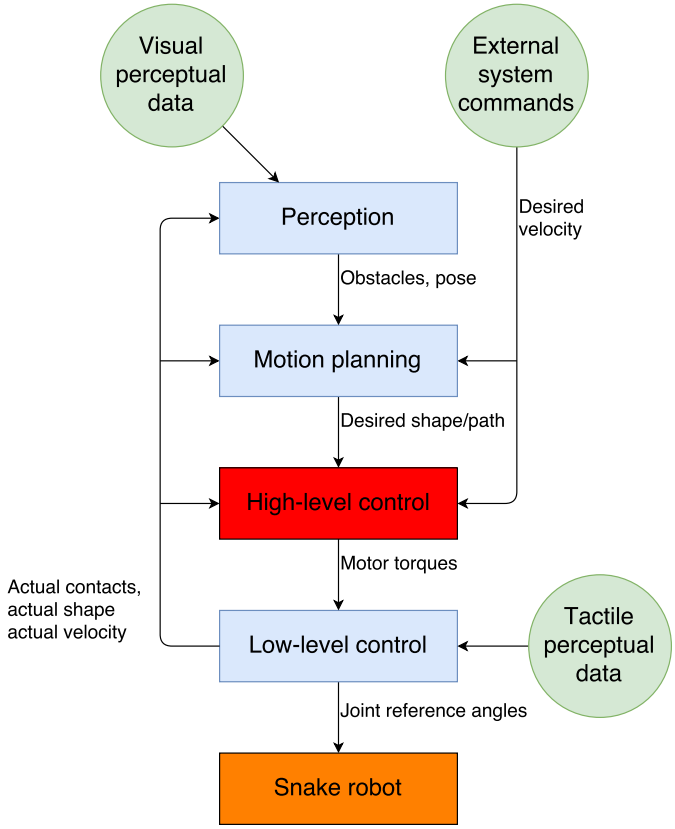


Figure 4.2: The hierarchical control framework for an autonomous snake robot.

In this thesis, perception and motion planning levels are assumed implemented, and the main focus is on implementing methods for high-level control.

Chapter 5

A control method for perception-driven obstacle-aided locomotion

5.1 Assumptions

The assumptions made about the snake robot and its surroundings are as follows:

1. The snake is planar.
2. There are $n \geq 3$ identical links.
3. Each pair of links are connected with an actuated joint.
4. The links and the obstacles are perfectly rigid, and the obstacles immovable.
5. The mathematical model is without friction, but the simulation is modelled with coulomb friction.
6. During locomotion the snake is in contact with at least three obstacles on alternating sides as described in Section 5.2.
7. Obstacles contact points are modelled as single points.

5.2 Propulsion method

Most of this section is a direct copy from my project report (Danielsen, 2016), but with the added subsection 5.2.1.

The equations of motion for the snake propulsion are based on (Sanfilippo et al., 2016b), and will be presented in this section for completeness. To ensure a well-formulated problem, these assumptions are made:

1. a path, S(s), with s being the path length parameter, is known. The obstacles locations, $\boldsymbol{o_1}$, $\boldsymbol{o_2}$, $\boldsymbol{o_3}$, are also known.
2. the snake is always on the path S(s).
3. the snake is planar.
4. the snake is continuous.
5. there is no ground or obstacles friction.
6. the snake is at rest
7. the snake tail link is tethered to the ground, as shown in Figure 5.1. The tether is unactuated. No tangential movements are allowed. The tail is not restricted in any other way. A tensile force, $\boldsymbol{f_s}$, acts along the tangent at $\boldsymbol{o_1}$.
8. the snake is perfectly rigid except at the point where an internal torque can be applied. The obstacles are perfectly rigid and fixed to the ground surface.
9. we choose to apply an internal torque, $\boldsymbol{\tau}$, at a known point, $\boldsymbol{p_{23}}$, on the path (i.e snake) between $\boldsymbol{o_2}$ and $\boldsymbol{o_3}$, as shown in Figure 5.1.

The forces working on the snake body are defined as follows:

$$\begin{aligned} \boldsymbol{f_s} &\triangleq |\boldsymbol{f_s}|(-\hat{\boldsymbol{t}}_{\boldsymbol{1}}), \\ \boldsymbol{f_1} &\triangleq |\boldsymbol{f_1}|(\hat{\boldsymbol{n}}_{\boldsymbol{1}}), \\ \boldsymbol{f_2} &\triangleq |\boldsymbol{f_2}|(\hat{\boldsymbol{n}}_{\boldsymbol{2}}), \\ \boldsymbol{f_3} &\triangleq |\boldsymbol{f_3}|(\hat{\boldsymbol{n}}_{\boldsymbol{3}}). \end{aligned} \tag{5.1}$$

The forces are shown in Figure 5.1. Since we assume to have knowledge about the position of the snake and the obstacles, we assume that the normal- and tangent vectors are known. This reduces the number of unknowns in the system to one per force vector, the unknowns now being the length of the vectors. We then have four unknowns.

It should be noted that $\boldsymbol{f_s}$ is a virtual force on the system. Solving the system therefore leads to an actual propulsion force working as $-\boldsymbol{f_s}$.

The torque working on a joint to right of the second obstacle can be mapped to $\boldsymbol{f_3}$ by the following equation:

$$\boldsymbol{f_3} = \boldsymbol{r} \times \boldsymbol{\tau} + \left[\frac{(\boldsymbol{\tau} \times \boldsymbol{r}) \cdot \hat{\boldsymbol{t}}_{\boldsymbol{3}}}{\frac{\boldsymbol{r}}{|\boldsymbol{r}|} \cdot \hat{\boldsymbol{t}}_{\boldsymbol{3}}} \right] \frac{\boldsymbol{r}}{|\boldsymbol{r}|}, \tag{5.2}$$

Where $\boldsymbol{r}$ is given as the vector from $\boldsymbol{p_{23}}$ to $\boldsymbol{o_3}$.

Since the torque is what we want to calculate, we must rewrite (5.2) to give $\boldsymbol{\tau}$ from $\boldsymbol{f_3}$ instead. We know by assumption that the snake robot is planar that $\tau_x = \tau_y = 0$, so we can expand the equation and solve for the scalar τ_z only. The expression we end up with is

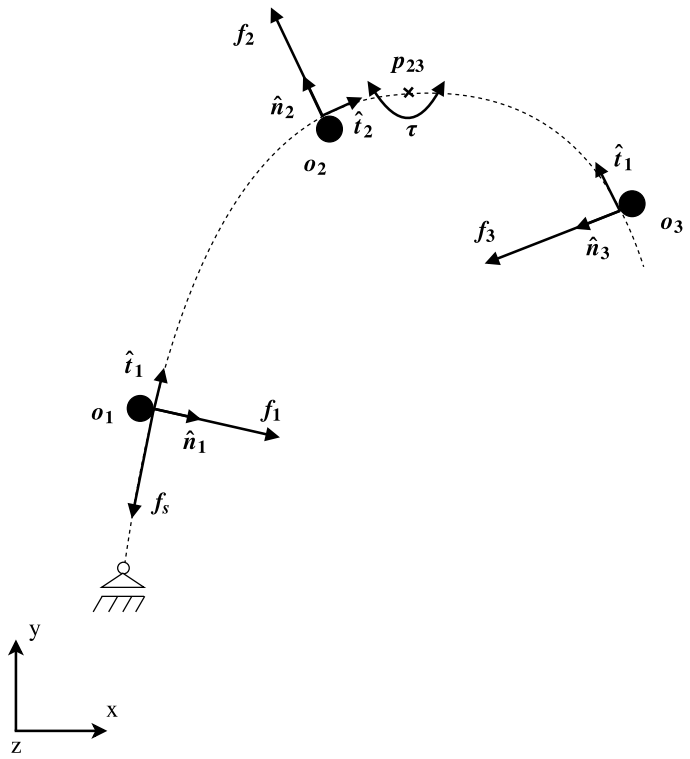


Figure 5.1: Obstacle triplet model. Each obstacle is marked as a black dot, and a the dashed line is a possible shape the snake can form around the obstacles.

$$\tau_z = \frac{|\boldsymbol{f_3}|\hat{n}_{3_x}(\boldsymbol{r} \cdot \hat{\boldsymbol{t}}_{\boldsymbol{3}})}{\hat{t}_{3_y}(r_x^2 + r_y^2)}. \tag{5.3}$$

This leaves only the problem of finding $|\boldsymbol{f_3}|$, in order to calculate the required propulsion torque. Assuming that $\boldsymbol{f_s}$ is given, and because we know the direction of all the forces, we are left with $|\boldsymbol{f_1}|$, $|\boldsymbol{f_2}|$, $|\boldsymbol{f_3}|$ as unknowns. To completely determine the system we need three equations for these three unknowns. Because of the assumption that the snake is at rest, we must have force balance:

$$\boldsymbol{f_s} + \boldsymbol{f_1} + \boldsymbol{f_2} + \boldsymbol{f_3} = \boldsymbol{0}. \tag{5.4}$$

As the snake is assumed to be planar, (5.4) gives us two equations. We need one more equation to determine the system. For the system to be at rest, we also need the torque working on the snake about the global origin to be zero. The equation for torque can be expressed as follows:

$$\boldsymbol{o_1} \times (\boldsymbol{f_s} + \boldsymbol{f_1}) + \boldsymbol{o_2} \times \boldsymbol{f_2} + \boldsymbol{o_3} \times \boldsymbol{f_3} = \boldsymbol{0}. \tag{5.5}$$

This equation is only relevant for z axis, and therefore gives one additional equation. This is as far as (Sanfilippo et al., 2016b) goes with the equations, and the continued derivations are done by this paper's author. To decompose the equations, we define the components of the normal- and tangents vectors as follows:

$$\hat{\boldsymbol{n}}_{\boldsymbol{i}} = \begin{bmatrix} \hat{n}_{i_x} \\ \hat{n}_{i_y} \\ 0 \end{bmatrix}, \hat{\boldsymbol{t}}_{\boldsymbol{i}} = \begin{bmatrix} \hat{t}_{i_x} \\ \hat{t}_{i_y} \\ 0 \end{bmatrix}, \boldsymbol{o_i} = \begin{bmatrix} o_{i_x} \\ o_{i_y} \\ 0 \end{bmatrix}, f_i = |\boldsymbol{f_i}|. \tag{5.6}$$

We split 5.4 into two equations, and expand (5.5) to get

$$f_s(-\hat{t}_{1_x}) + f_1\hat{n}_{1_x} + f_2\hat{n}_{2_x} + f_3\hat{n}_{3_x} = 0, \tag{5.7}$$

$$f_s(-\hat{t}_{1_y}) + f_1\hat{n}_{1_y} + f_2\hat{n}_{2_y} + f_3\hat{n}_{3_y} = 0, \tag{5.8}$$

and

$$-\begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{t}_{1_x} & \hat{t}_{1_y} \end{vmatrix} f_s + \begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{n}_{1_x} & \hat{n}_{1_y} \end{vmatrix} f_1 + \begin{vmatrix} o_{2_x} & o_{2_y} \\ \hat{n}_{2_x} & \hat{n}_{2_y} \end{vmatrix} f_2 + \begin{vmatrix} o_{3_x} & o_{3_y} \\ \hat{n}_{3_x} & \hat{n}_{3_y} \end{vmatrix} f_3 = 0. \tag{5.9}$$

Solving the system of (5.7), (5.8) and (5.9) for f_3, we get a solution on the form

$$f_3 = \alpha f_s, \tag{5.10}$$

where α is a scalar depending on the geometry, given by

$$\alpha = -\frac{\begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{n}_{1_x} & \hat{n}_{1_y} \end{vmatrix} \frac{\hat{t}_{1_x}}{\hat{n}_{1_x}} - \begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{t}_{1_x} & \hat{t}_{1_y} \end{vmatrix} + \left(\begin{vmatrix} o_{2_x} & o_{2_y} \\ \hat{n}_{2_x} & \hat{n}_{2_y} \end{vmatrix} - \begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{n}_{1_x} & \hat{n}_{1_y} \end{vmatrix} \frac{\hat{n}_{2_x}}{\hat{n}_{1_x}} \right) \frac{\hat{t}_{1_y} - \frac{\hat{n}_{1_y}}{\hat{n}_{1_x}}\hat{t}_{1_x}}{\hat{n}_{2_y} - \frac{\hat{n}_{1_y}}{\hat{n}_{1_x}}\hat{n}_{2_x}}}{\begin{vmatrix} o_{3_x} & o_{3_y} \\ \hat{n}_{3_x} & \hat{n}_{3_y} \end{vmatrix} - \begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{n}_{1_x} & \hat{n}_{1_y} \end{vmatrix} \frac{\hat{n}_{3_x}}{\hat{n}_{1_x}} + \left(\begin{vmatrix} o_{2_x} & o_{2_y} \\ \hat{n}_{2_x} & \hat{n}_{2_y} \end{vmatrix} - \begin{vmatrix} o_{1_x} & o_{1_y} \\ \hat{n}_{1_x} & \hat{n}_{1_y} \end{vmatrix} \frac{\hat{n}_{2_x}}{\hat{n}_{1_x}} \right) \frac{\frac{\hat{n}_{1_y}}{\hat{n}_{1_x}}\hat{n}_{3_x} - \hat{n}_{3_y}}{\hat{n}_{2_y} - \frac{\hat{n}_{1_y}}{\hat{n}_{1_x}}\hat{n}_{2_x}}}.$$

Now we have all the equations we need, as (5.10) and (5.3) gives the relationship between the desired propulsion force f_s and the required propulsion torque $\boldsymbol{\tau}$. f_s is supposed to be given by V_d, but for the experiment presented here, a constant f_s was used.

In the implementation of the controller, the vector $\boldsymbol{r}$ was changed to be defined as $\boldsymbol{r} = \boldsymbol{o_3} - \boldsymbol{o_2}$. This is expected to give more stable values, although it compromises the accuracy of the calculated torque τ_z.

The propulsion controller is a ROS node that subscribes to three topics that give obstacle positions, robot pose and contact point data, and desired propulsion speed. These topics together give all the data needed for the equations above. After doing the calculations, the node publishes on two topics, one for desired torque, and one for which joint to apply the torque.

Here ends the copy from (Danielsen, 2016).

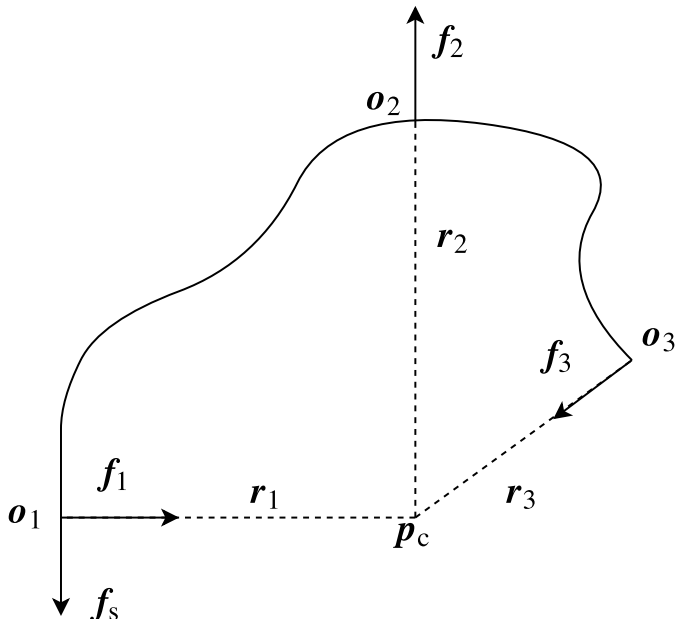


Figure 5.2: The condition in which no propulsion force is produced.

5.2.1 Geometric conditions

Here a geometric condition that will prevent the snake robot from gaining propulsion is presented.

From the equations of motion we can find that there is a certain geometrical configuration that will make any propulsion impossible. In this configuration the lines drawn along the normals of the robot links from the contact points all intersect in one point $\boldsymbol{p}_c$, as in Figure 5.2. The vectors from the contact points $\boldsymbol{o}_1$, $\boldsymbol{o}_2$, $\boldsymbol{o}_3$ to $\boldsymbol{p}_c$ are represented by $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, $\boldsymbol{r}_3$. Since these vectors are parallell to their corresponding contact forces $\boldsymbol{f}_1$, $\boldsymbol{f}_2$, $\boldsymbol{f}_3$ respectively, they will not contribute to any torque relative to the point $\boldsymbol{p}_c$. Torque balance around $\boldsymbol{p}_c$ then yields $\boldsymbol{f}_s \times \boldsymbol{r}_1 = 0 \leftrightarrow \boldsymbol{f}_s = 0 \forall \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3$. Since the tensile force, $\boldsymbol{f}_s$, then is null, there will not be produced any propulsive force in this geometric configuration. Care should therefore be taken to avoid the snake robot getting stuck in this position.

5.3 Perception of environment

For the propulsion controller to work, information about the snake robot's surroundings is required. The perception of the environment is done through sensors placed on the robot. The controller module needs to interact with these sensors through the interfaces implemented for them. These interfaces were implemented in this work and are presented in Section 6.

The information about the environment needed by the propulsion controller is:

- Contact points for three obstacles on alternating sides
- Normal- and tangent vectors for the contact points
- The position of the joint applying torque relative to the third contact point

Chapter 6

Implementation

6.1 Sensor nodes

In this section all the implemented ROS nodes which function as sensors, i.e the nodes that output some form of information about the state of the system, are presented.

6.1.1 Robot pose module

One piece of information that we needed was the position and orientation of all the links of the snake robot. This information was already supplied by the node */gazebo* over the topic */gazebo/link_states*, but orientations were described with quaternions which needed to be converted to euler angles. Since three nodes required this information (and maybe more in the future), we decided to make a new node that would pre-process the link state information and output it on a format that could be used directly in the receiving nodes. The new node subscribed to the */gazebo/link_states* topic, and published a new formatted message on the */snakebot/robot_pose* topic. The new message contained the following information:

- link names
- link numbers
- link poses in 2D

The pose of a link was described by the values of x, y, θ, describing the position and orientation with respect to the world frame. An illustration of the message content for a single link can be seen in Figure 6.1.

6.1.2 Collisions module

As aforementioned, a bumper sensor interface node was first implemented by Guillame (Adam, 2016). This interface was limited in usability, as it was designed to only work

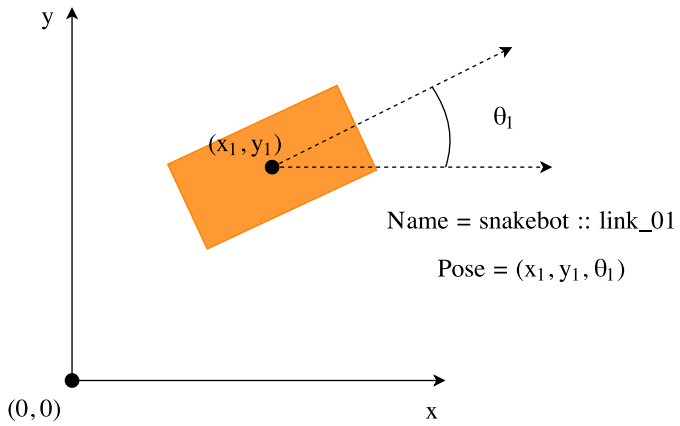


Figure 6.1: Example of pose data for one link, as given by the Robot Pose module. The name of the link is "snakebot::link_01", the position is (x_1, y_1), and it's rotated θ_1 radians with respect to the x-axis.

with the original obstacle triplet model with obstacles on alternating sides. If the number of obstacles was different from three, or if the obstacles were not on alternating sides, the node would simply not publish any messages. As we wanted to make a more complete and functional system that could be developed and improved without continuously rewriting old code, this had to be changed. In an effort to generalize the module to work for an arbitrary number of obstacles, it was rewritten based on the basic functionality from the previous version. An overview of the design of the new module for processing collision data is presented here.

Gazebo offered several sensors as plugins for ROS. For obtaining collision and contact force data, we used the sensor called *Bumper*. A bumper sensor was included in the ROS description of each link of the snake robot. By doing this, Gazebo automatically created a publisher for each sensor, which published the available data for the bumper on the bumper's own topic. It was then possible to subscribe to these topics to get the data we needed in other nodes. The messages sent from each sensor was of the message type *gazebo_msgs/ContactsState*. This message again contained an array of messages of the type *gazebo_msgs/ContactState*, where each message was for one pair of contact elements involving that specific bumper sensor. Let us take a look at the information provided for each pair of elements in contact. The data (excluding metadata) given by the sensor was:

- Names of the two links colliding
- Forces and torques
- Total forces and torques
- Contact positions
- Contact normal vectors

- Penetration depth of the collision

From Section 5.2 we see that there is one piece of information about the contact point that is missing in the above list. That is the contact point tangent vectors. Since the direction of the tangent vector was defined relative to the forward direction of the snake robot link, more information about the link was needed to determine the direction of the vector. This extra information could be easily obtained from the robot pose node, as it gave the orientation of each link. Using contact normal vector from the bumper sensor in addition to the orientation of the link, we could find the tangent vector. The approach was simple; the one tangent of our two possible ones that was within 90 degrees from the direction vector of the link was our desired tangent. We had $\boldsymbol{t} = \boldsymbol{t}_i$, where $\boldsymbol{t}_i$ is one of the possible tangents with $i \in \{1, 2\}$, if $\alpha < 90°$, where α is the angle between $\boldsymbol{t}_i$ and the directional vector $\boldsymbol{d}$ of the robot link. We could most likely have used a smaller angle than 90°, but if the obstacles collide with the corners of the robot links, it's not easy to know at what angle the tangent vectors will point. An illustration of the approach is shown in Figure 6.2.

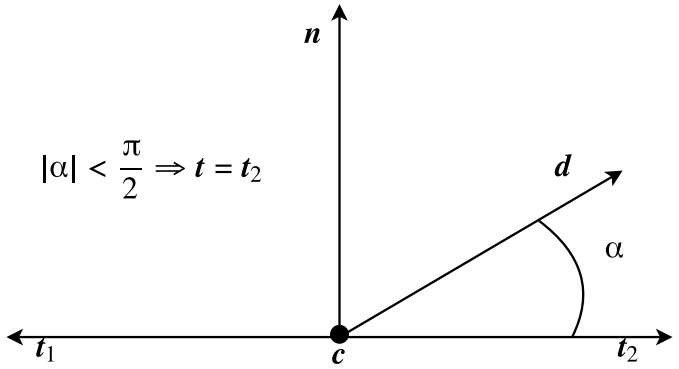


Figure 6.2: How to find the correct tangent. $\boldsymbol{n}$ is the normal vector arising from the collision at point $\boldsymbol{c}$. The tangent $\boldsymbol{t}$ should be within 90 degrees of the directional vector $\boldsymbol{d}$ of the robot link. The correct tangent is in this case the alternative $\boldsymbol{t}_2$.

It is also necessary to know if the obstacles are on the right or the left side of the snake robot in order to determine which obstacles to use in the propulsion method. This can easily be found using the normal and tangent vectors. We know that

$$\hat{\boldsymbol{n}}_{\boldsymbol{i}} \times \hat{\boldsymbol{t}}_{\boldsymbol{i}} = \begin{bmatrix} 0 \\ 0 \\ \pm 1 \end{bmatrix}, \tag{6.1}$$

where the sign of the third component indicates which side the obstacle is on. A positive sign means that the obstacle is on the left side of the snake when moving from the tail, and vice versa for a negative sign. An illustration is shown in Figure 6.3.

Torques, total forces and torques, and penetration depth were not used, as they were not needed for our purpose. To make sure that multiple collisions between one snake robot link an obstacle was not counted as one, the total force vector was not used. But since the

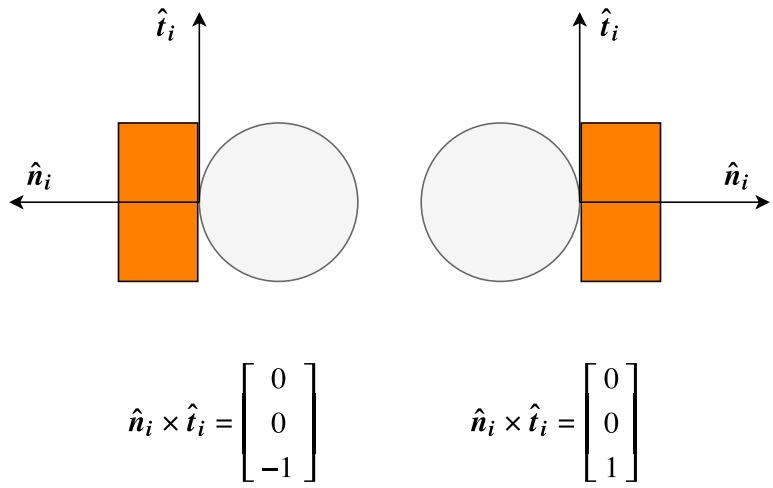


Figure 6.3: Showing the relationship between the tangent- and normal vectors given by which side of the snake the obstacle is on. The orange rectangles are snake links, and the gray cylinders are obstacles.

theoretical propulsion model was in 2 dimensions only, contact points lying on top of each other were counted as one. After the collision data had been processed, the information available for each collision through the published messages was:

- The number of the snake robot link involved in the collision
- Which side of the robot the contact is on (left or right)
- Normal vector for the contact point (pointing towards the robot link)
- Tangent vector for the contact point (pointing forward)
- Position of the contact point

A visual explanation of the data given by the obstacle interface can be seen in Figures 6.4 and 6.5. The interface publishes a ROS message containing this information for all the collisions. Figure 6.6 shows an overview of the communication between the collisions node and the rest of the system.

6.1.3 Push-point module

Now that we had the obstacle collision data available, we wanted to extract the data needed by the propulsion controller. The theoretical model used for caclulating propulsion uses exactly three obstacles on alternating sides. Three obstacles in this configuration needed to be identified and sent to the propulsion controller. We therefore decided to make another module, a push-point node, for easy access to the required data. This module starts from the tail of the snake, and finds the three first obstacles in the correct configuration, while

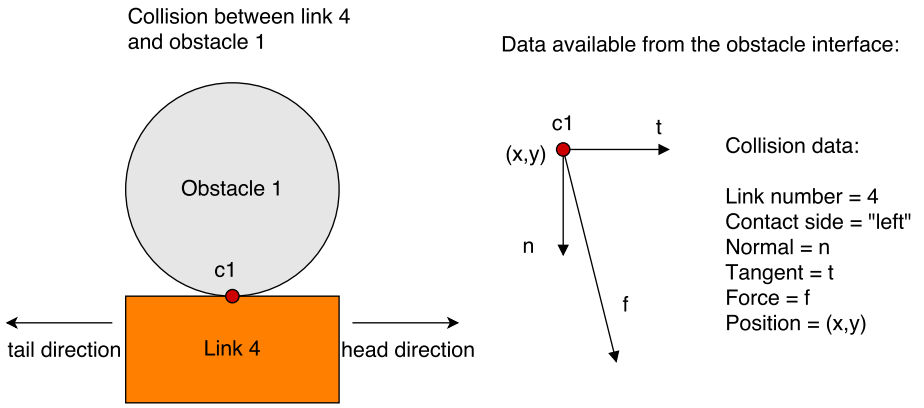


Figure 6.4: Left: A visual representation of the collision between Obstacle 1 and Link 4 in 2D. Right: The data from this collision avilable through the obstacle interface.

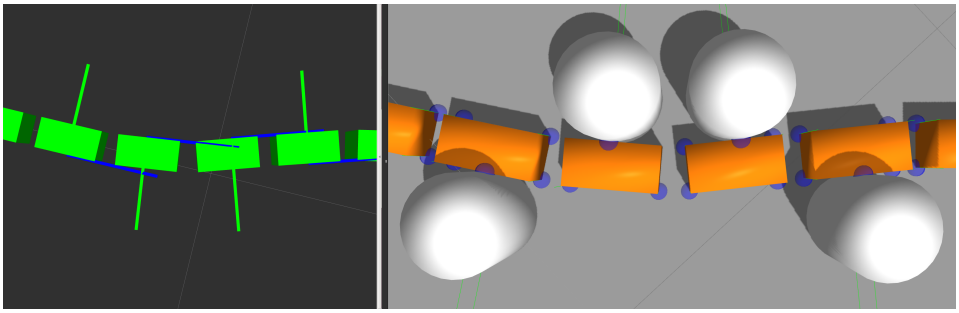


Figure 6.5: Right: Gazebo showing all contact points as blue spheres. Left: RViz displaying the normal- and tangent vectors given by the collisions module.

ignoring all the other collisions. We then get the same data as from the obstacle interface, but ordered as collision 1, 2 and 3 starting from the tail end of the snake robot. Figure 6.7 shows three examples of how the module would choose which contact points to use as push-points.

6.2 Controllers

6.2.1 Propulsion controller

The implementation of the propulsion controller was not changed much since my project (Danielsen, 2016), but an overview of the functionality is presented here.

The main algorithm is an implementation of the propulsion method presented in Section 5.2. It subscribes to the the topic */snakebot/pushpoints* to get data on the obstacle triplet, and to */snakebot/propulsion_force* to get the desired forward propulsion force. The controller chooses the first joint that comes after the middle contact point as the actuator

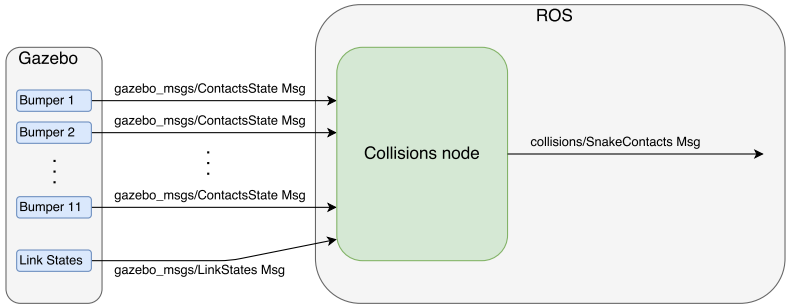


Figure 6.6: Overview of the messages sent to and from the obstacle interface.

for propulsion. The push-point data is the run through the propulsion algorithm to get the torque output needed to gain the desired forward propulsion force. The propulsion joint number and torque is then published on the topic */snakebot/propulsion_effort*.

Two small modifications to the propulsion controller was done in this work. First, the controller was set to only output torque that would lead to a push on the third obstacle in the triplet model. This keeps snake robot from pulling away from the obstacle when it's in a configuration that leads to a calculated torque in the wrong direction. Secondly, the duration of continued operation after losing contact with obstacles was increased. This helped make sure the snake robot would get back in contact with the obstacle if some unexpected movement had pushed it away, although it also prevented the robot from stopping smoothly after moving past the final obstacle.

Propulsion joint

In this work, only a single and arbitrary joint between the second and third contact point was chosen to generate the necessary torque to gain propulsion. We chose the first joint that comes after the second contact point, but there might be better options. A possible approach to find a better suited joint for torque generation is to calculate the required torque for several candidate joints, and compare them to find which one requires the least amount of torque. An even bigger improvement would be to distribute the torque on all of the joints between the second and third obstacles. This could be achieved using a hybrid angle/torque controller, similar to the hybrid position/force controller in (Raibert and J., 1981). With this hybrid controller we could make sure that robot stays on its path by keeping the joints the desired angles, while at the same time generating enough torque to propel itself forward along the obstacles.

6.2.2 Keeping contact with the obstacles

The main problem with the previous propulsion method in (Danielsen, 2016) was that the robot kept losing contact with obstacles, making the forward locomotion come to a halt. Two methods for keeping the robot pushing on the obstacles were implemented.

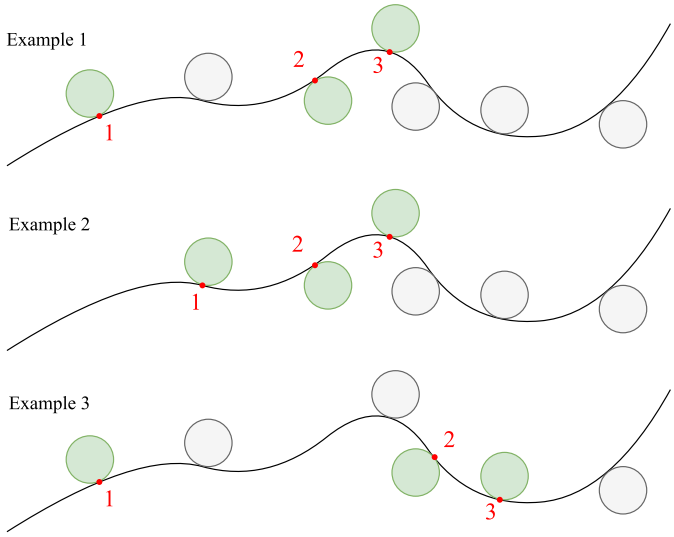


Figure 6.7: The push-point module selects the first alternate-side obstacle triplet it can find, starting from the earliest possible contact point.

6.2.3 Spring mode controller

The first method we tried for keeping the snake robot in contact with obstacles was to make the robot behave in a spring-like manner, inspired by the flexible metal band shown in Figure 6.8. This was done by setting the joint angle references to be constant 0, and using the joint angle controller to adhere to this reference. This in effect turned the snake robot into a spring always wanting to be straight. If the snake robot was then pushed into a curve by the surrounding obstacles it would push back against them, exerting enough force for the bumper sensors to work. This mode could be effective in a dense obstacle environment, as the robot would passively reach for a new obstacle as soon as contact with the previous obstacle was lost. A demonstration video of the spring mode controller was made, see Appendix.

There was however a few of drawbacks to this approach. Firstly, with three obstacles the snake robot would only be able to curve in one direction, depending on the placement of the obstacles. Secondly, the forces exerted by the snake robot would increase as it became more curved, possibly interfering with the propulsion controller. The third and biggest problem was the symmetry around the middle obstacle that tended to arise when the snake robot was in contact with only three obstacles. The normal vectors at the contact points were often close to intersecting at the same point, as described in Section 5.2.1. Because of these problems, we looked for another approach.

6.2.4 Shape compliance controller

The biggest contribution to propulsion method was through the implementation of a shape compliance controller. This controller was designed to help put the snake robot in a config-

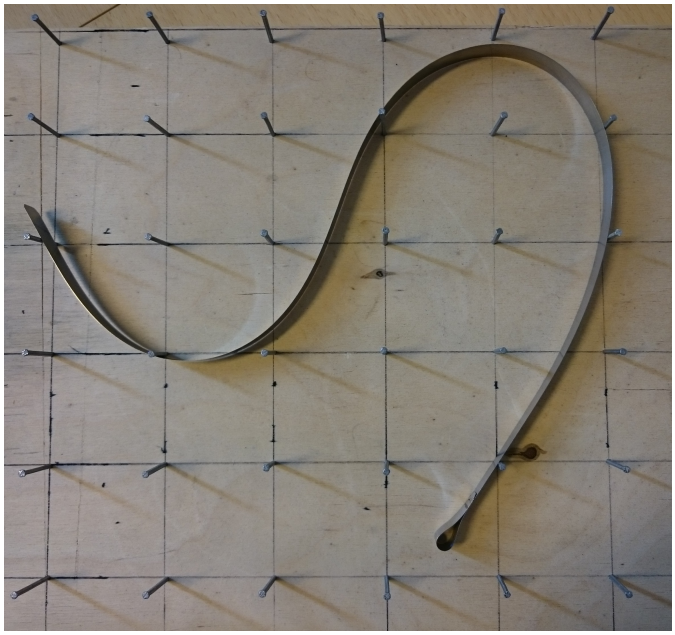


Figure 6.8: The spring mode was implemented to achieve a behavior similar to that of the flexible metal band in this picture.

uration that give rise to forward propulsion being generated by the propulsion controller. Let's take a look at this module.

The idea is that from a desired path $P(s)$, actuator inputs must be generated to keep the robot on the specified path. The path is made up of several points, as shown in Figure 6.9. A general motion planning framework for body shape control of snake robots is presented in (Liljebäck et al., 2014). Based on the theory in that paper, a shape compliance controller was implemented.

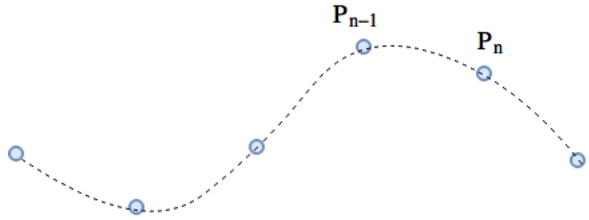


Figure 6.9: The desired path of the snake is specified through discrete points.

A simplification done in (Liljebäck et al., 2014) is that the links of the snake robot are modelled as straight lines. An actual snake robot, and the one that's used in the simulation described in this thesis, has a thickness that will not allow the center of the links to be placed alongside objects. This has to be accounted for in an implementation of the path

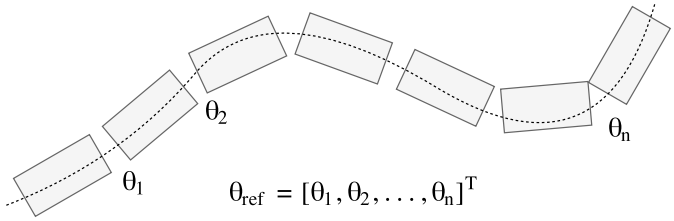


Figure 6.10: The virtual snake robot is placed on the path, and its joint angles are used as reference values for the controlled snake robot.

following algorithm.

As our point of interest was obstacle-aided locomotion, and we wanted the snake robot stay in contact with obstacles, the control points for the shape was set to be contact points in the obstacle triplet model. The idea was to use the fact that the thickness of the snake robot links would make the robot deviate a little from it's desired path, making it continue to exert enough force on the obstacles to get data from the bumper sensors.

In order to place the snake robot on the desired path, we employ the usage of a so-called "virtual snake robot", as described in (Liljebäck et al., 2014). The joint angles needed for the virtual snake robot to be on the path is then sent as reference angles to the real snake robot, as illustrated in Figure 6.10.

It should be noted that the shape controller starts by running spring mode control until it finds three contact points.

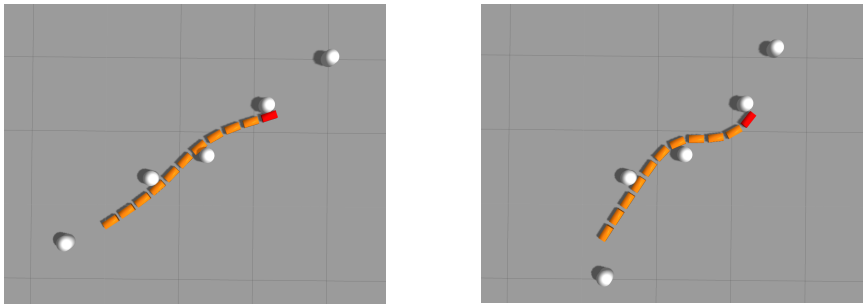
Straight line segments

What we ended up doing was instead of having the robot trying to adhere to a straight line like a spring, straight line segments was defined between each pair of obstacles. One advantage of straight line segments between obstacle compared to the single straight line, is that we can remain in contact independent of the geometry of the obstacle points. A problem that still remained with this approach was that we could not specify a configuration which facilitated propulsion generation, as the angle of attack on the obstacles were constant given one set of three obstacles. We needed more flexibility in specifying the shape of the snake robot.

Path generator

Since the straight line segments approach was not good enough either, a more flexible path generator was implemented. The idea was that the path generator both could be used to specify a path between the obstacles that was smooth, and to attain a desired contact point geometry.

The control points for generating the shape curve was set to be the contact points for the obstacles in the triplet configuration. In a addition to these, two extra control points were used, one behind the tail and one in front of the head of the snake. The virtual points were added to be able to adjust the angle of attack on the first and third obstacle in the triplet configuration. The shape curve was then generated by interpolating between



(a) Configuration 1 **(b)** Configuration 2

Figure 6.11: Here you can see how moving the control points on the obstacles in front and behind the snake robot changes the shape curve which the robot follows.

these setpoints. We ended up using a quadratic spline interpolation method, as it produces smooth curves similar to trigonometric interpolation, and because an open source library for this was readily available. The interpolation was done with the y-position as a function of the x-position in the world frame, making the method only work for set-points with increasing value on the x-axis.

To be able to change the shape of the curve while running the simulation, the front and rear control points were set to be at the center of two obstacle cylinders not in contact with the robot. During the simulation, these two cylinders could be moved around using the GUI in Gazebo to change the shape curve. Figure 6.11 shows how moving the two obstacles affects the shape of the snake robot.

6.2.5 Joint angle controller

To be able to used the shape compliance controller, a controller for the joint angles was needed. The controller that was used in ROS was from the package *ros_control*, and of the type *effort_controllers/JointEffortController*. The controller used torque as input, so a joint angle controller with a torque control signal was needed. This was done by implementing a simple PD controller. This type of controller was used for quick design, and to ensure fast and stable control. The reason a PID controller was not used is that the joints may have constant deviations from the reference because of obstacles in the way. This would make the integrator wind up, which could lead to problems with instability. The control signal is given by the following equation:

$$\boldsymbol{\tau_d} = \boldsymbol{K_p}(\boldsymbol{\theta} - \boldsymbol{\theta_d}) + \boldsymbol{K_d}(\dot{\boldsymbol{\theta}}),$$

where $\boldsymbol{\theta}$, $\boldsymbol{\theta_d}$ and $\dot{\boldsymbol{\theta}}$ are the joint angles, desired joint angles and joint angular velocities respectively. The derivative of the joint angle, $\dot{\boldsymbol{\theta}}$, was used instead of the standard $\dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_{\boldsymbol{d}}$ because it was given directly from the joint controller, eliminating the need for differentiating $\dot{\boldsymbol{\theta}}$ and $\dot{\boldsymbol{\theta}}_{\boldsymbol{d}}$ which might contain high frequency components.

6.2.6 Torque selector

This section is copied from (Danielsen, 2016).

The joint angle controller (also called position controller) generates torques for all joints, while the propulsion controller generates desired torque for one joint. These two signals must be combined to one shared unit in order to get a consistent low level controller input. The concept of a hybrid position/force controller (Raibert and J., 1981) was used as an inspiration to make this. The hybrid controller that is described is used for applying forces to an end effector of a robot manipulator, while adhering to both physical and virtual constraints. For the case of the snake robot, the controlled value is the torques for all the joints on the robot. This was achieved by using a simple torque selector. The torque selector gets values from both the propulsion controller and the position controller, as seen in Figure 6.12. From the propulsion controller it gets a desired torque and the joint number to apply it on. This joint is then controlled for propulsion, while all the others are controlled by the input from the position controller. Combining the two reference signals, a single torque reference signal τ_d is generated and sent to the robot low level controller.

Figure 6.13 shows the state diagram for each joint switching between being controlled for propulsion and position.

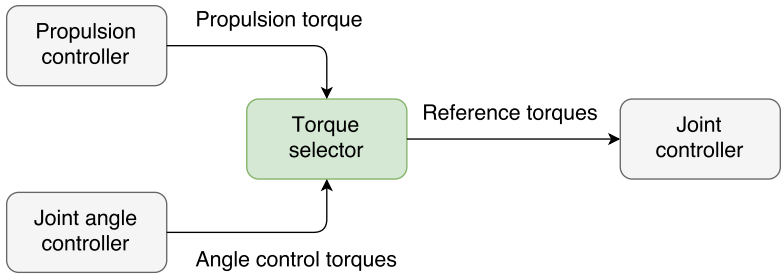


Figure 6.12: The torque selector gets input from the propulsion controller and the joint angle controller. It selects which of the received torques to use, and outputs the resulting vector of reference torques to the joint controller.

The torque selector was implemented as a ROS node. It subscribes to three topics, one for desired joints torques for position control, one for desired torque on one joint for propulsion control, and one for which joint to apply the propulsion torque. Based the data from these topics, the controller publishes on topics for the desired effort of the joint motors. The controller for the motors is the *ros_control* node.

Here ends the copy from (Danielsen, 2016).

6.3 Simulation description

As explained before, Gazebo was used to carry out the simulations of the controller implemented on the robot model. The physical attributes were all specified in a robot description file for ROS, which was then loaded into the simulator when launching Gazebo. The snake robot consisted of 11 identical links, each of the dimension 20 x 10 x 10 cm^3 and weighing

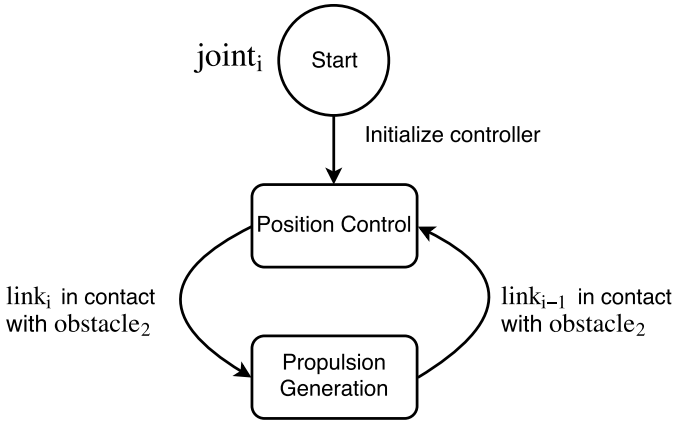


Figure 6.13: State diagram showing how each joint switches between being controlled for propulsion and position. $obstacle_2$ is the middle obstacle in the triplet model.

1 kg. The links were space 6 cm apart, connected by rotational actuated joints, rotating about the z-axis. In the robot descriptions for ROS and Gazebo, a joint can be placed anywhere as a singular point, so it was placed mid-air in the middle of the space separating the links. In the description file, one could set certain parameter values for the joints. We set max effort to 100, max velocity to 1, damping to 100, and friction to 0. For the robot links we tried to set different coulomb friction coefficient for the long side and the short side, but what seemed to be an unresolved issue in this version of Gazebo made the friction be set for the x- and y-axis in the world frame instead. Because of that problem we ended up setting the friction coefficient to be the same in all directions. We wanted a low friction in the forward sliding direction, so a friction coefficient of 0.3 was used after experimenting with different values. Illustrations of the links and obstacles can be seen in Figures 6.14 and 6.15 respectively.

The physics engine ODE was used for the simulations.

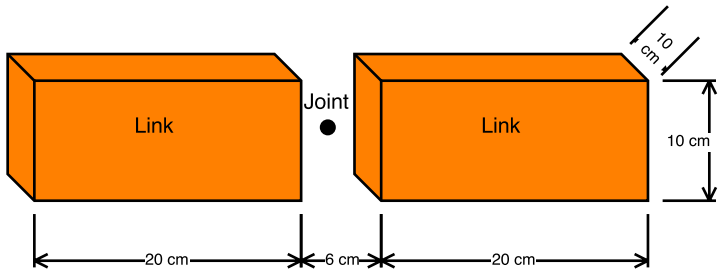


Figure 6.14: Illustration of the link size and their connection. The joint only rotates around the z-axis.

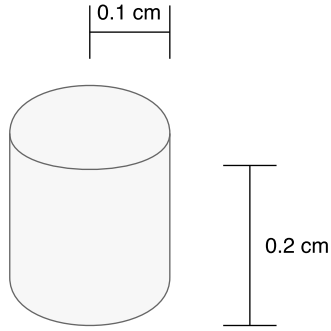


Figure 6.15: Illustration of the obstacle shape and size.

6.4 Overview of modules

A sequence diagram for one control iteration is shown in figure 6.16. Note that the ROS gui tool *rqt_publisher* was used to set the desired forward propulsion force.

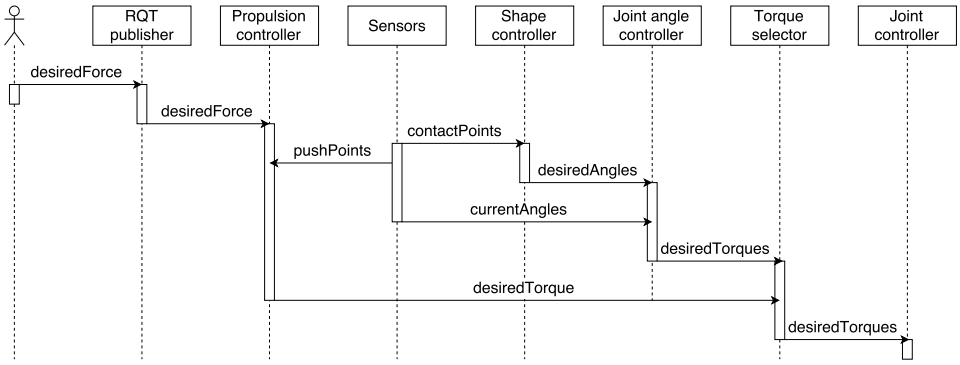


Figure 6.16: A diagram showing one iteration of the control sequence. The diagram is simplified by removing callback functions to avoid clutter, and the propulsion controller and joint angle controller are in reality run in parallel, as opposed to in sequence as portrayed by this diagram.

In Figure 6.17 we see a diagram showing the communication between the ROS nodes. The names in the figure are not all exactly the same as in the implementation, and several of the same type of topic have been combined to single topics to make the graph cleaner.

An overview of the framework can be seen in Figure 6.18. From this figure, it's easy to understand the idea of later replacing Gazebo with a real snake robot. Since RViz is a ROS tool, it can be used either with a simulator like Gazebo, or with a physical robot for visualizing forces and torques affecting the robot.

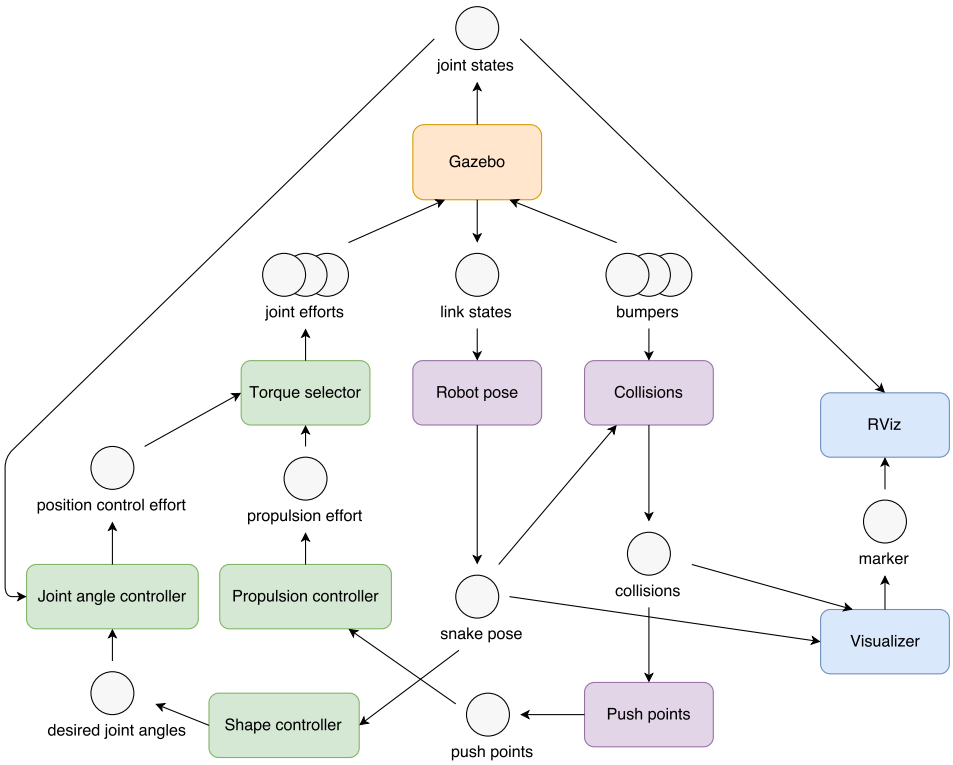


Figure 6.17: A diagram showing the communication between the ROS nodes when running propulsion control in the simulated environment. The colored boxes are nodes, and the circles are topics. Stacked circles represents a group of topics with the same function. Green nodes are generate control signals, while purple nodes acts as sensors or processors of sensor data. The blue node are for visualization. Gazebo, the orange node on the top, contains the low level controllers and sensors.

6.5 Visualization

In order to better understand the propulsion movement a node for visualizing the forces working on the snake robot was implemented. This was done using the RViz visualization tool for ROS. The displayed data was mainly related to the data coming from the bumper sensors in contact with the obstacles, but also a vector showing the value of the calculated torque for propulsion. At the points of contact on the snake robot, vectors showing the normal, tangent and forces were displayed. The sum of the contact forces were also displayed.

6.6 Communication between ROS and LabVIEW

As a step towards realizing this controller on a physical robot, communication between the snake robot interface and ROS had to be established. The interface for the snake robot was

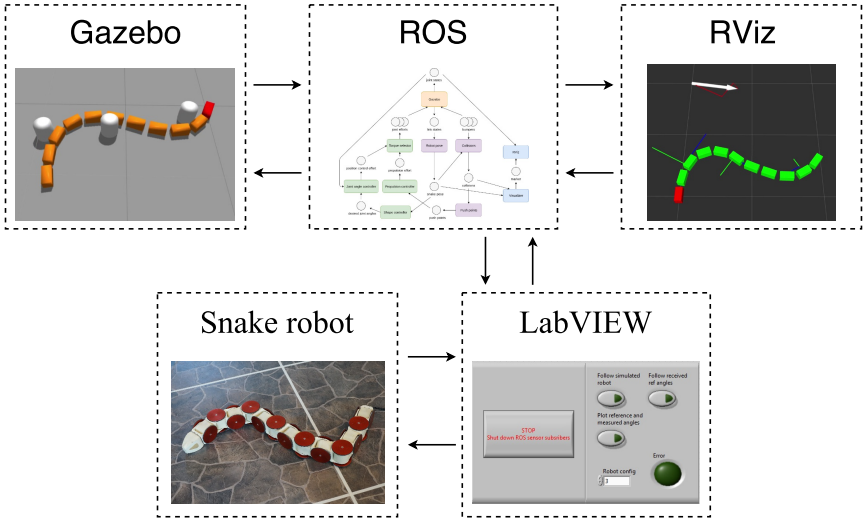


Figure 6.18: An overview of the framework. Gazebo and RViz are both ROS nodes in this case, but are put outside as they can be replaced by a real snake robot and a different interface software respectively.

implemented in LabVIEW. Because of this, we needed to find a way for our ROS nodes to communicate with LabVIEW. Luckily, we found ROSforLabVIEW, the software package introduced in Section 2.5, which we could used exactly this purpose.

We wanted to build a framework to make it possible for researchers to implement controllers in both ROS and LabVIEW, and to connect to the physical robot and/or the simulated one, according to their needs. This was done by setting up a ROS node responsible for the communication between the two sides. This node was made to read all available sensor data from each side, and transmit it over to the opposite sides. It can also transmit control data, so that LabVIEW can send control signals to ROS and vice versa. Figure 6.19 shows how the controller for the simulated robot can be implemented on the LabVIEW side, and Figure 6.20 shows how the controller for the real snake robot can be implemented in ROS. Note that the controllers and sensors in ROS were not implemented on the LabVIEW side in this work.

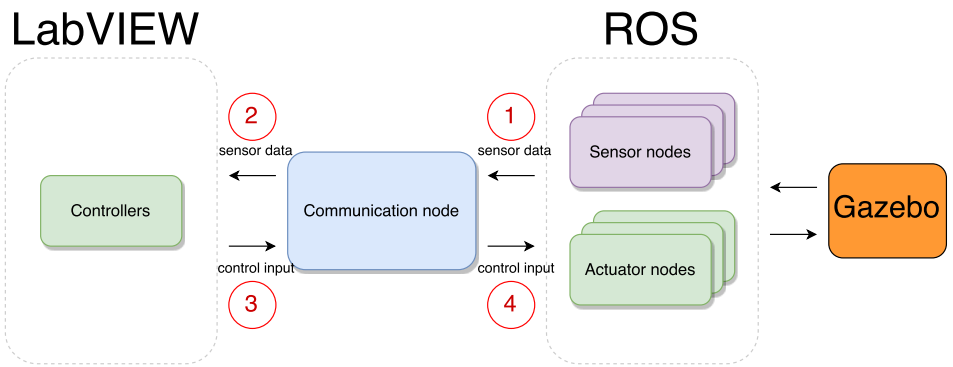


Figure 6.19: A controller for the simulated robot can be implemented in LabVIEW.

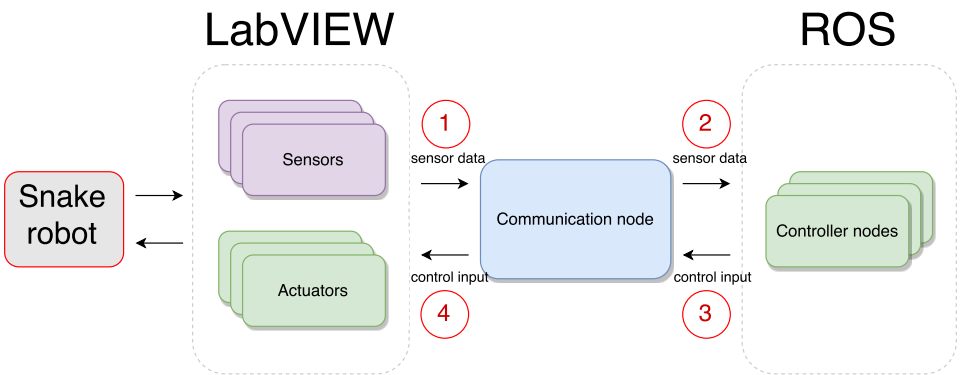


Figure 6.20: A controller for the real robot can be implemented in ROS.

Chapter 7

Simulations

7.1 Propulsion method

In this chapter we first present how the simulation of the propulsion method was carried out. Then we present the results obtained from the simulation. A discussion of the results can be found in Chapter 9.

7.2 Initialization

Before loading the robot into the simulator, we needed to load a world. We made a world file were we specified the physical descriptions and starting position of all the obstacles we wanted to have in our simulation. To spawn the robot, we needed to start a *spawn_model* node from the package *gazebo_ros*. The node takes in the ROS robot description file as a parameter, and spawns the robot in Gazebo. Additional optional parameters for the spawner are inital joint angles, which can be used to set the starting pose of the robot. The set the inital positon of the snake robot to be in between the obstacles, we set the starting joint positions to be:

$$\boldsymbol{\theta_{start}} = \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 \end{bmatrix}^T.$$

Figure 7.1 shows initial configuration of the snake right after spawning. Going into spring mode control from the starting configuration guaranteed three push-points, and testing showed that the shape controller could also maintain contacts when starting this way.

Before starting the propulsion phase, the snake robot was moved into its starting pose by the shape controller. This was done by placing the two movable control points (obstacle cylinders) in a way the made the snake form a shape that ensured contact with the obstacles and a push-point geometry suitable for propulsion. The starting configuration can be seen in Figure 7.2.

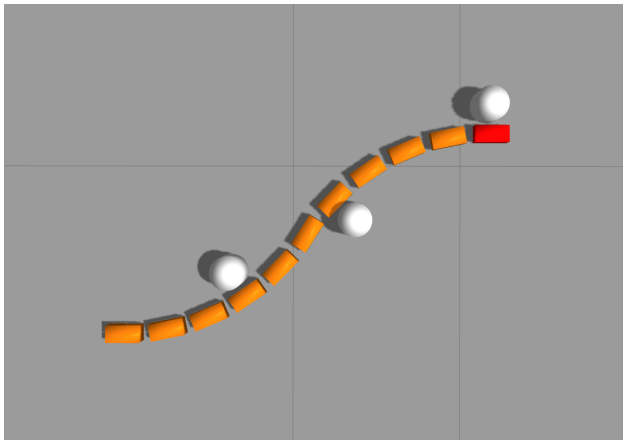


Figure 7.1: The initial state of the snake after spawning it in Gazebo. The head of the snake is colored red.

7.3 Running the simulation

The human operator should ideally be able to specify a forward propulsion velocity. An acceleration a would be found from this velocity, and the propulsion force would be found from the acceleration using Newtons' second law, $f = ma$. This conversion had not yet been implemented, so a constant force was used instead.

For this simulation a desired propulsion force $f_s = 80$ N was chosen. A high force was chosen as the robot sometimes stopped with a lower value when the obstacles went between two links.

The sudden onset of propulsion torque makes the snake move away from the obstacles, losing contact data for propulsion control. A temporary fix for this was to let the propulsion controller use old contact data for a duration long enough for the robot to regain contact with the obstacles. In the experiment, a duration of 1 second could pass after losing contact before the controller stopped outputting torque. This also helped the robot to stay on the path, as the shape controller could not always keep the robot in contact with the obstacles.

7.4 Results

A demonstration video for the propulsion method was made, see Appendix.

Figures 7.3 and 7.4 show the joint angles and applied joint torques respectively for all the 10 joints during the propulsion phase. The first joint to be controlled in propulsion mode was joint 7, as link 7 started in contact with the middle obstacle. This can also be seen in both torque- and angle plots for the joint, as there is a sudden high torque value at the same time angle suddenly deviates far from the reference value. As the snake robot continues to move forward, the transfer of propulsion torque to the next joints 6, 5 and 4 can be seen observed the same way.

The joints furthest from the propulsion joint can be seen to follow the reference better

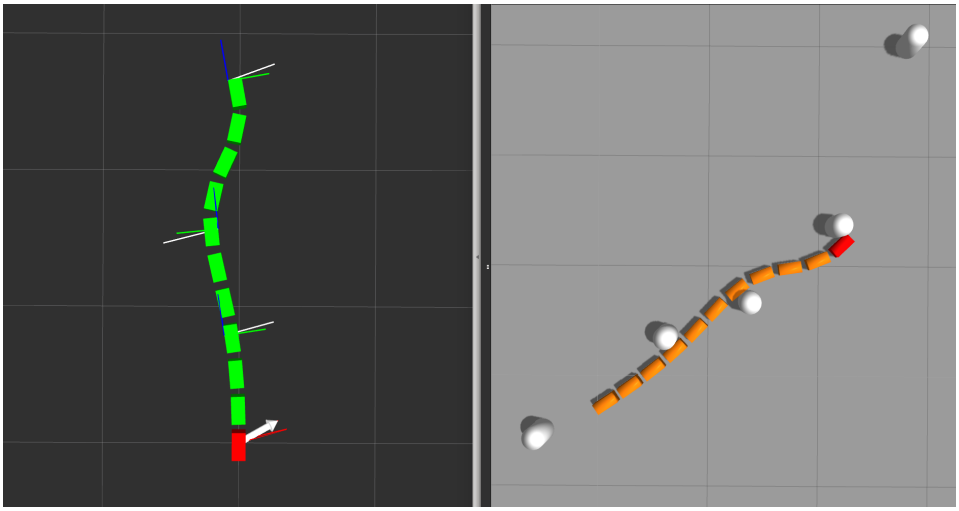


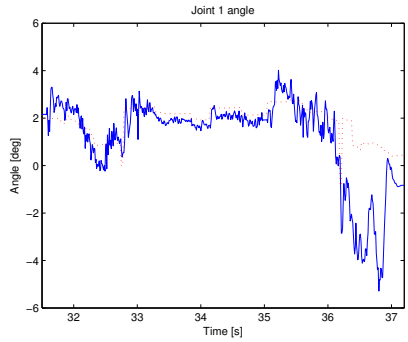
Figure 7.2: The starting configuration before running the propulsion experiment. From RViz on the left we can see that the snake robot receives contact data from all three obstacles.

than the rest, as we can see that joint 10 follows the reference better over time, while joint 1 becomes worse. The same can be seen more clearly for joint 9 and 2 which are closer to the propulsion generating joint.

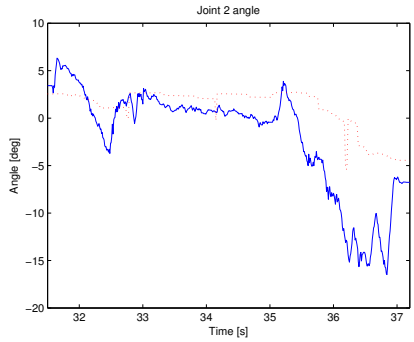
There are some visible peaks in the reference angles for all the joints, and corresponding peaks can also be seen in the torque plots. It is not immediately obvious where these originated from.

As is true for both measured joint angles and applied torque, there is a lot of rapid change in the values during the whole propulsion phase.

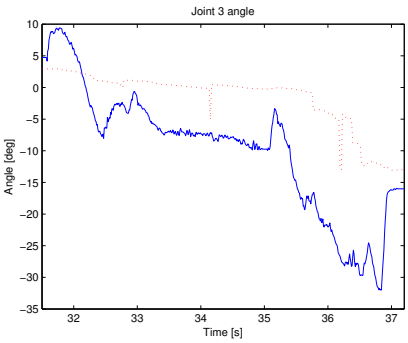
During the propulsion phase, sudden movements which move the snake robot away from the front obstacle can be observed. This is happening at the time of transitioning the propulsion generation control to the next joint as the robot moves forward. Figure 7.5 shows the change in position of the snake as the torque generation task is passed on to the next joint.



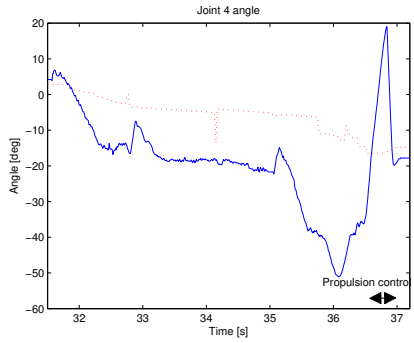
(a) Joint 1



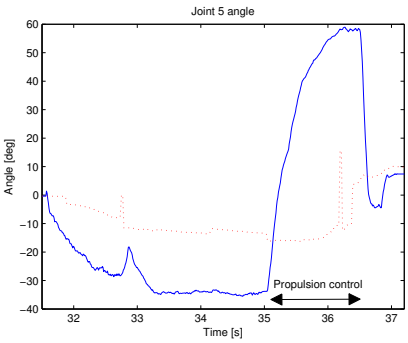
(b) Joint 2



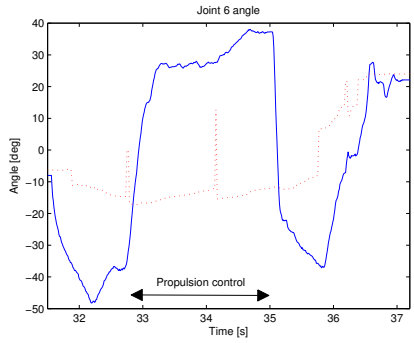
(c) Joint 3



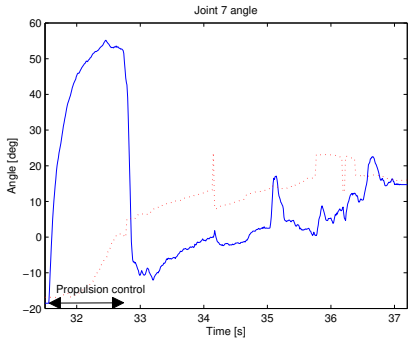
(d) Joint 4



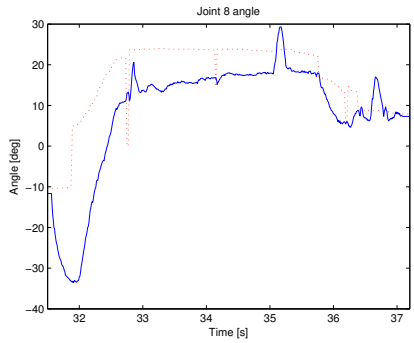
(e) Joint 5



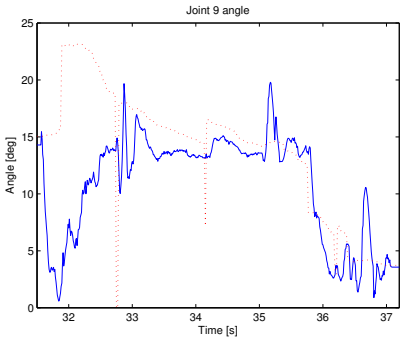
(f) Joint 6



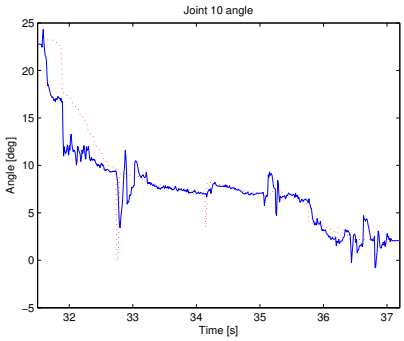
(g) Joint 7



(h) Joint 8

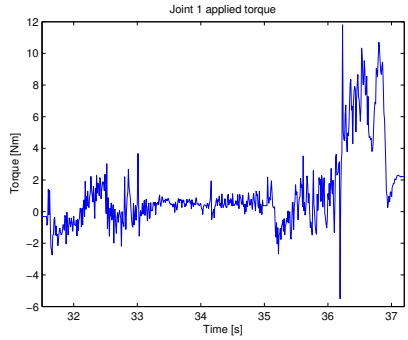


(i) Joint 9

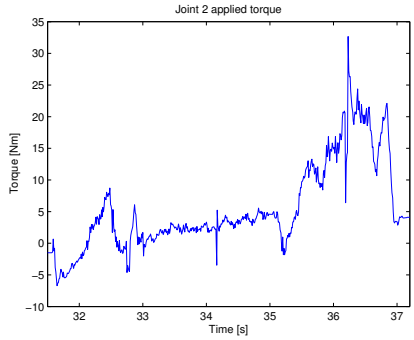


(j) Joint 10

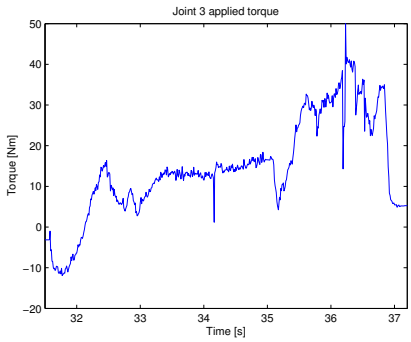
Figure 7.3: Joint angles for all 10 joints during the propulsion phase. The dashed red lines are the reference angles given by the shape controller, and the blue lines are the measured angles. The two-sided arrows in the plots show the approximate time that specific joint is controlled for propulsion generation. The rest of the time the joints are in angle control mode.



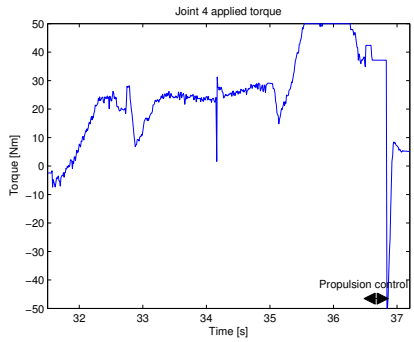
(a) Joint 1



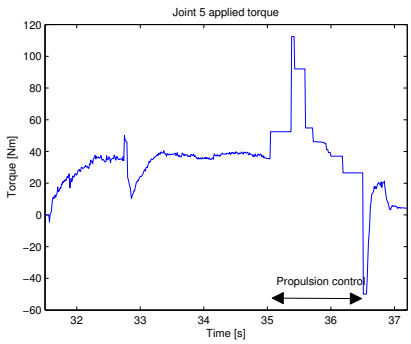
(b) Joint 2



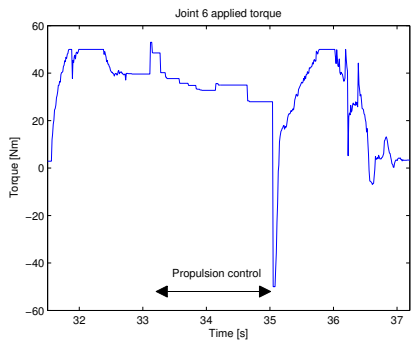
(c) Joint 3



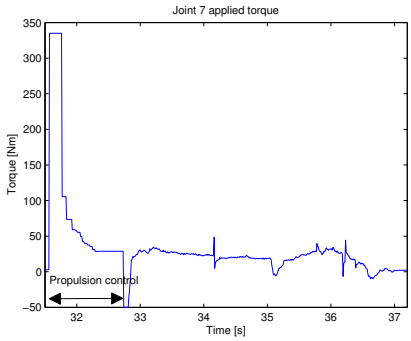
(d) Joint 4



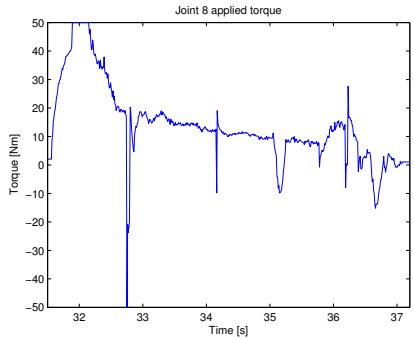
(e) Joint 5



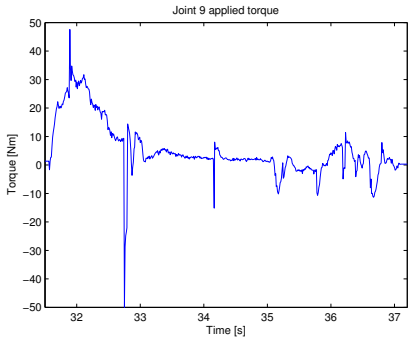
(f) Joint 6



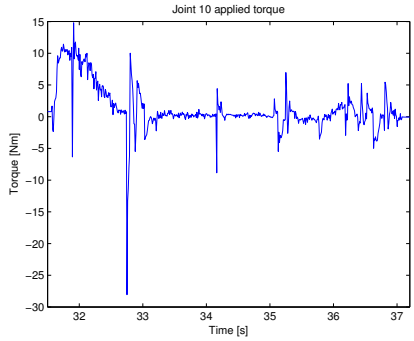
(g) Joint 7



(h) Joint 8

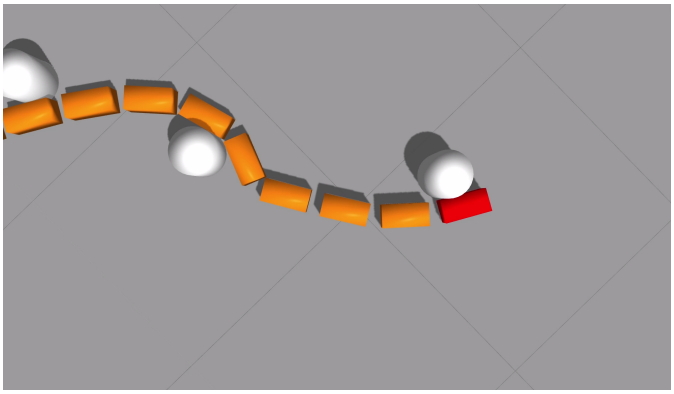


(i) Joint 9

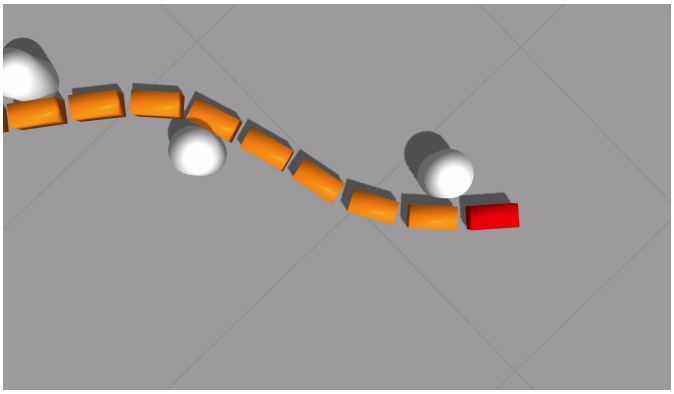


(j) Joint 10

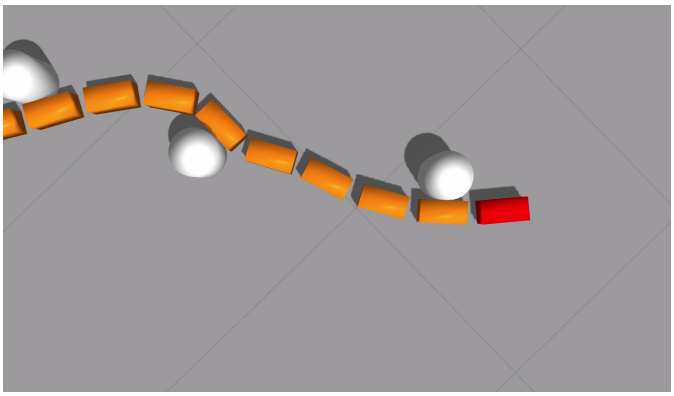
Figure 7.4: Joint torques for all 10 joints during the propulsion phase. The two-sided arrows in the plots show the approximate time that specific joint is controlled for propulsion generation. The rest of the time the joints are in angle control mode.



(a) Right before the transition.



(b) Transition has just occurred, and contact is lost.



(c) After reacquiring contact with the obstacle.

Figure 7.5: A sudden position change at the time of switching joint for propulsion generation.

Chapter 8

Experiments

8.1 Communication Interface

To test the functionality of the communication interface, two experiments were run to test the connection both ways. For the first experiment the real snake robot was set to follow the same movement as the simulated one by using its measured joint angles as reference values. The second experiment was the same, only the other way around. A discussion of the results can be found in Section 9.

8.2 Mapping between simulated and real snake robot

The simulated snake robot had 11 links and 10 actuated joints in yaw configuration, while the physical robot had 13 links with only 6 joints controlled in yaw. In order to test the communication between the to frameworks, in our case ROS and LabVIEW, we wanted to see if the physical robot was able to follow the movement of the simulated robot and vice versa. Because the number of joints were different on the two robots, we had to make a mapping from one set of joints to the other in order get the desired result as in Figure 8.1. Three approaches were considered for this purpose. The first was to just remodel the simulated snake robot to have as many joints as the physical version, and then copy the measured joint angles in the simulation to reference values for the physical robot. This way we would actually remove the need for a mapping. A second possibility was to employ a synergy control method. The use of postural synergies for controlling a grasping robotic hands is discussed in (Prattichizzo et al., 2010). This method could be used to map the overall posture of the snake robots, instead of each individual joint, and use that to generate new reference angles for the other robot to match this posture. The third approach that was considered was to generate a shape curve based on the pose of the snake robot, and then map that curve into joint angles for the follower robot.

We decided to go with the postural synergies approach, as it can easily be modified to be used with other robots with different joint configurations, allowing researchers to work

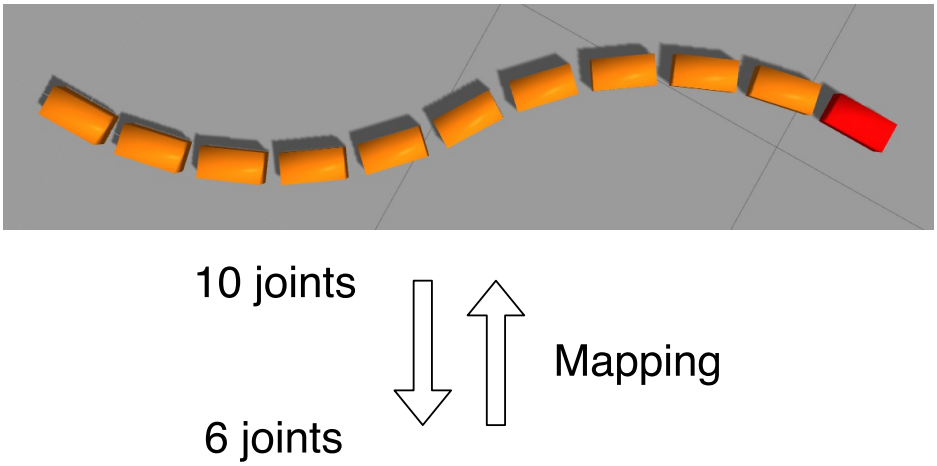


Figure 8.1: Desired mapping of posture from one robot to the other. Note that we are only using the planar joints of the real snake robot.

with simulation models in the same or different configurations than the real robots. It was also easier to implement than the shape curve method. A mapping from m to n joints was done by using a transformation matrix that distributed the m joint angles evenly over n joints.

The mapping equation is given by

$$\boldsymbol{M}\boldsymbol{\theta}_a = \boldsymbol{\theta}_b, \tag{8.1}$$

where M is a $n \times m$ mapping matrix, $\boldsymbol{\theta}_a$ is a $m \times 1$ column vector containing joint angles of snake robot a, $\boldsymbol{\theta}_b$ is a $n \times 1$ column vector containing joint angles needed for a snake robot b to match the posture of snake robot a.

8.2.1 The mapping matrix

Here an approach to designing a mapping matrix is presented.

Ideally we would have wanted to follow an approach similar to the one in (Prattichizzo et al., 2010), by finding the main components that constitutes snake movement and finding parameters for mapping movement from one robot to the other. Since there hasn't been done any experiments to identify the parameters for this mapping, we decided to do a linear combination of the joints to achieve a result that was accurate enough for our purpose.

Each column of the mapping matrix specifies how much of a joint angle of robot 1 should add to a joint angle in robot 2. As long as the sum of the elements in a column vector equals 1, we know that the curvature of the corresponding joint in robot 1 has been transferred to robot 2. When the sum of the column elements equal 1 for all the columns, the total curvature of robot 1 will be transferred to robot 2, and the heads of the two snake robots will have the same orientation relative to their tails.

To make sure that the curvature induced by a joint in robot 1 resulted in the same curvature around the same point on robot 2, we used normal distributions to make sure that most of the angle value of a joint was mapped to the closest joints in the other robot. The standard deviation of the normal distribution was set relatively low, to avoid distributing the angles outwards too much, which would given a smoothing effect on the posture curve. The values of the normal curves were places on the columns of the mapping matrix. If necessary, the columns were then scaled to attain an element sum closer to 1. A graphical representation of the distribution used to map 10 joint angles to 6 can be seen in Figure 8.2.

For mapping the 10 joints on the simulated robot to the 6 joints on the real robot, we used the mapping matrix defined by

$$\boldsymbol{M_{10to6}} = \begin{bmatrix} 0,84 & 0,44 & 0,10 & 0,01 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,17 & 0,52 & 0,70 & 0,35 & 0,06 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0,020 & 0,20 & 0,60 & 0,66 & 0,27 & 0,04 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0,04 & 0,27 & 0,66 & 0,60 & 0,20 & 0,02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0,06 & 0,35 & 0,70 & 0,52 & 0,17 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0,01 & 0,10 & 0,44 & 0,84 \end{bmatrix}$$

For mapping the 6 joints on the simulated robot to the 10 joints on the real robot, we used the mapping matrix defined by

$$\boldsymbol{M_{6to10}} = \begin{bmatrix} 0,62 & 0,06 & 0 & 0 & 0 & 0 \\ 0,33 & 0,30 & 0,01 & 0 & 0 & 0 \\ 0,05 & 0,43 & 0,09 & 0 & 0 & 0 \\ 0 & 0,18 & 0,35 & 0,01 & 0 & 0 \\ 0 & 0,02 & 0,40 & 0,13 & 0 & 0 \\ 0 & 0 & 0,13 & 0,40 & 0,02 & 0 \\ 0 & 0 & 0,01 & 0,36 & 0,18 & 0 \\ 0 & 0 & 0 & 0,09 & 0,43 & 0,05 \\ 0 & 0 & 0 & 0,01 & 0,30 & 0,33 \\ 0 & 0 & 0 & 0 & 0,06 & 0,62 \end{bmatrix}$$

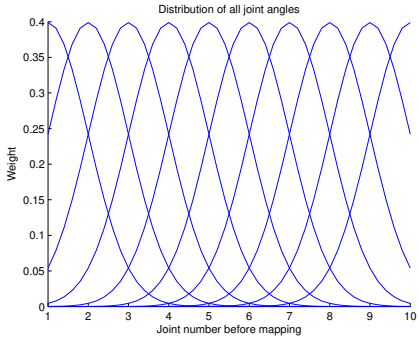
When comparing the two matrices, it can be seen that the sum of the row elements for each row in $\boldsymbol{M_{10to6}}$ is higher than 1, while the sums are less than 1 for the rows in $\boldsymbol{M_{6to10}}$. This is due to the fact we are compressing data when mapping to a configuration with a lower degree of freedom, and expanding data for the opposite mapping.

8.3 Sequence diagrams

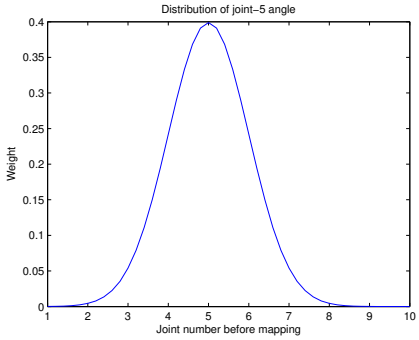
Figures 8.3 and 8.4 show the sequence diagrams for one iteration of the control sequence for the two experiments.

8.4 Results

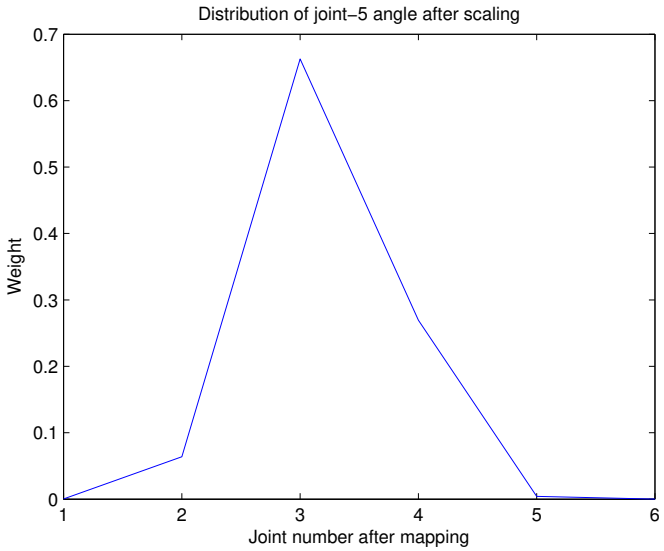
The main results for these experiments can be seen in the demonstration video showing how the simulated and real robots could follow each others movements. To find the video, see Appendix.



(a) Distribution of all angles



(b) Distribution of the angle of joint 5



(c) The distribution weight values for the mapped joints from joint 5 after scaling. This values can be found in the column 5 of the mapping matrix M from 10 to 6 joints. The lack of symmetry is due to the fact that joint 3 in the new mapping is not aligned with joint 5 in the original mapping.

Figure 8.2: The distribution method used to map 10 joint angles to 6.

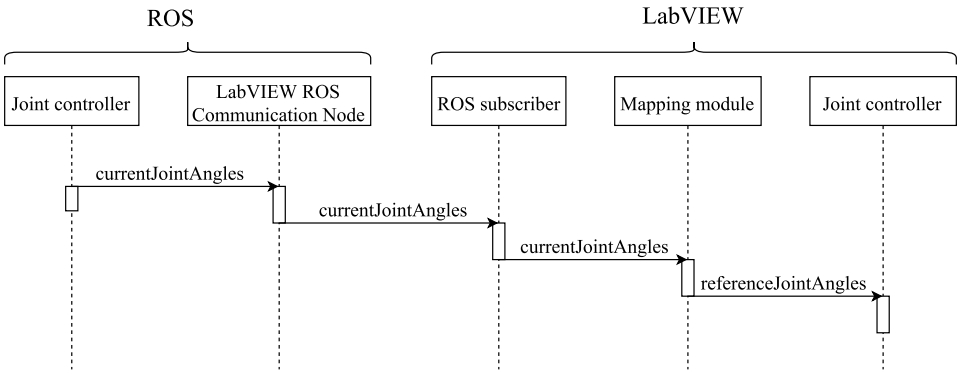


Figure 8.3: Sequence diagram showing the communication between ROS and LabVIEW for one iteration when the real snake robot follows the simulated snake robot.

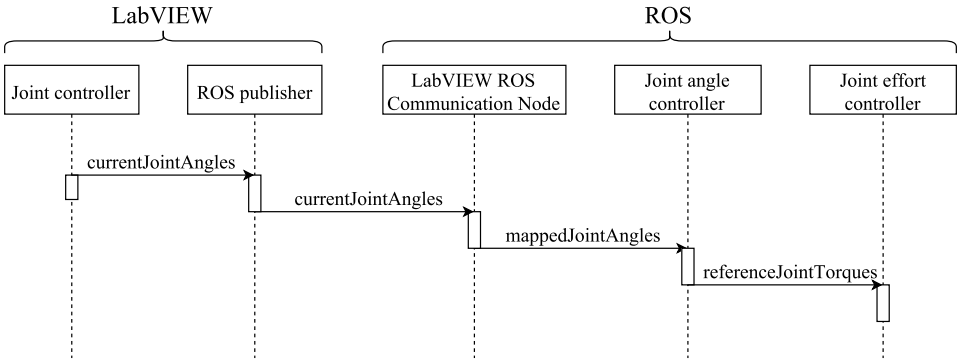


Figure 8.4: Sequence diagram showing the communication between ROS and LabVIEW for one iteration when the simulated snake robot follows the real snake robot.

Chapter 9

Discussion

9.1 Propulsion method

The propulsion method managed to get the snake robot to move forward a length of almost four links. Compared to the previous results in (Danielsen, 2016), where the locomotion stopped after the first link, this was an improvement. The contributions from this thesis that lead to this improvement were the implementation of a more generalized collision sensor interface, and more importantly the shape controller. The new collision interface made sure that the propulsion controller kept getting useful data, even when two links were in contact with the same obstacle at the same time. The shape controller allowed the snake to follow the path it was moving on, letting the the robot slide along with the movement, instead of staying rigid and resisting movement.

There is however room for more improvement on the propulsion method. As could be seen from the plots of joint angles and applied torques, the movement was not very smooth, with unnecessary movement and deviation from the desired path. In Figures 7.5a and 7.5c, a visibly sharper angle appears at the joint responsible for propulsion generation. This is undesired, as there is a sudden large change in angle, as in Figure 7.5b, when it goes back to angle control mode. What appears to be high frequency noise in plots might be the result of joint angle controllers with not enough damping.

The shape controller is at this point not good enough either, as it can't guarantee contact with the obstacles.

9.2 Future work on the propulsion method

9.2.1 Velocity and direction controller

We want the user of the system to be able to input desired velocity and movement direction to the snake robot. An input interface for this was implemented by (Adam, 2016), but there is currently no method for converting these inputs to control values for the propulsion controller.

9.2.2 Push-points module

The selection of obstacle triplet could be optimized. The current module makes no consideration of the which push-points would be the best choice. It also only finds one obstacle triplet, even though there could be many. For a future implementation we want to identify which triplet or triplets can give the highest potential of generated forward propulsion.

9.2.3 Friction

Friction should be modelled and considered for a more realistic control method.

9.2.4 Path generator

As the current path generator doesn't work in all directions in the plane, it should be improved. The new path generator should be able to generate a path between any given setpoints in the plane, independent of the direction between the points. As the perception system gets completed, and visual information is available, the movable set-point in front of the snake could be the next desired obstacle.

9.2.5 Path follower

A path following controller should be properly implemented. Its purpose would be to improve on the current shape controller, to guarantee that the robot actually stays in the contact with obstacles required for generating propulsion, while following the path as the snake robot moves forward.

9.2.6 Hybrid force/position controller

Something that we would like to implement in future works is a proper hybrid position/-force controller. This could be based on the controller described in (Raibert and J., 1981). Using such a controller, one could make sure that both natural and virtual constraints on force and position for the snake robot would be met during the propulsion phase. All the joints could be controlled for angular position and propulsion generation simultaneously. This should in theory allow for a much smoother and elegant forward motion.

9.3 Communication interface

The current implementation of the communication interface allows the two snake robots to copy each others movement. This is just a demonstration to show that the interface works, and that it can be further developed to achieve more useful functionality.

9.4 Future work on the communication

The potential of communication between the simulated and real robots was explored, but further work needs to be done in order to use for perception-driven obstacle-aided loco-

motion.

The first thing that needs to be done is to port all the functions implemented in ROS to LabVIEW. This includes especially the propulsion controller, but also methods for shape control and processing contact sensor data.

The real snake robot does not support torque control for the joint motors, so some method for achieving controlled torque needs to implemented. There are strain gauges on the joints which can be used to measure torque, but these needs to be calibrated before they can be used. If the strain gauges are calibrated properly, it should be possible to make a torque controller.

Chapter 10

Conclusion

The main tasks for this thesis were to improve on a previous implementation of a control method for perception-driven obstacle-aided locomotion for snake robots, implement a communication interface between the ROS framework and the interface for the real snake robot, and lastly to preform simulations and experiments to validate these implementations.

The control method was improved by upgrading previously implemented modules, and by implementing a shape controller to help guide the robot forward. The improvements could be seen through simulations, but there is still work remaining before a robust and accurate control method for perception-driven obstacle-aided locomotion is realized.

A communication interface was made, and tested for two scenarios through experiments. The foundation for the communication is there, and now it remains to implement similar functionality on both sides in order to realize perception-driven obstacle-aided locomotion on a real snake robot.

Bibliography

Adam, G., 2016. User and simulation interface for the snake robots perception-driven obstacle-aided locomotion. Student project report, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Engineering Cybernetics, Trondheim, Norway.

Andruska, A. M., Peterson, K. S., 2008. Control of a snake-like robot in an elastically deformable channel. IEEE/ASME Transactions on Mechatronics 13 (2), 219–227.

Bayraktaroglu, Z. Y., Blazevic, P., 2005. Understanding snakelike locomotion through a novel push-point approach. Journal of dynamic systems, measurement, and control 127 (1), 146–152.

Bayraktaroglu, Z. Y., Kilicarslan, A., Kuzucu, A., Hugel, V., Blazevic, P., 2006. Design and control of biologically inspired wheel-less snake-like robot. In: The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. IEEE, pp. 1001–1006.

Danielsen, S. G., 2016. A control method for perception-driven obstacle-aided locomotion of snake robots. Student project report, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Engineering Cybernetics, Trondheim, Norway.

Gray, J., 1946. The mechanism of locomotion in snakes. Journal of Experimental Biology 23 (2), 101–120.

Gupta, A., 2007. Lateral undulation of a snake-like robot. Ph.D. thesis, Massachusetts Institute of Technology.

Hirose, S., Mori, M., 2004. Biologically inspired snake-like robots. In: Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on. IEEE, pp. 1–7.

Holden, C., Stavdahl, Ø., Gravdahl, J. T., 2014a. Optimal dynamic force mapping for obstacle-aided locomotion in 2d snake robots. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 321–328.

Holden, C., Stavdahl, Ø., Gravdahl, J. T., 2014b. Optimal dynamic force mapping for obstacle-aided locomotion in 2d snake robots. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 321–328.

Kamegawa, T., Kuroki, R., Gofuku, A., 2014. Evaluation of snake robot's behavior using randomized earli in crowded obstacles. In: 2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014). IEEE, pp. 1–6.

Kamegawa, T., Kuroki, R., Travers, M., Choset, H., 2012. Proposal of earli for the snake robot's obstacle aided locomotion. In: Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on. IEEE, pp. 1–6.

Lee, M. C., Park, M. G., 2003. Artificial potential field based path planning for mobile robots using a virtual obstacle concept. In: Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on. Vol. 2. IEEE, pp. 735–740.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., Gravdahl, J. T., 2014. Compliant control of the body shape of snake robots. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, pp. 4548–4555.

Ma, S., Ohmameuda, Y., Inoue, K., 2004. Dynamic analysis of 3-dimensional snake robots. In: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Vol. 1. IEEE, pp. 767–772.

Nogueira, L., 2014. Comparative analysis between gazebo and v-rep robotic simulators, unpublished Manuscript.

Nor, N. M., Ma, S., 2014. Cpg-based locomotion control of a snake-like robot for obstacle avoidance. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 347–352.

Perrow, M. R., Davy, A. J., 2008. Handbook of Ecological Restoration: Volume 1, Principles of Restoration. Cambridge University Press.

Prattichizzo, D., Malvezzi, M., Bicchi, A., 2010. On motion and force control of grasping hands with postural synergies.

Raibert, M. H., J., C. J., Jun. 1981. Hybrid position/force control of manipulators. Journal of Dynamic Systems, Measurement, and Control (103.2), 126–133.

Sanfilippo, F., Azpiazu, J., Marafioti, G., Transeth, A. A., Stavdahl, Ø., Liljebäck, P., 2016a. A review on perception-driven obstacle-aided locomotion for snake robots. In: Proc. of the 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand.

Sanfilippo, F., Stavdahl, Ø., Marafioti, G., Transeth, A. A., Liljebäck, P., 2016b. Virtual functional segmentation of snake robots for perception-driven obstacle-aided locomotion. In: Proceeding of the IEEE Conference on Robotics and Biomimetics (ROBIO), Qingdao, China. IEEE.

Shan, Y., Koren, Y., 1993. Design and motion planning of a mechanical snake. IEEE transactions on systems, man, and cybernetics 23 (4), 1091–1100.

Shan, Y., Koren, Y., 1995. Obstacle accommodation motion planning. IEEE Transactions on Robotics and Automation 11 (1), 36–49.

Transeth, A. A., Leine, R. I., Glocker, C., Pettersen, K. Y., Liljebäck, P., 2008. Snake robot obstacle-aided locomotion: Modeling, simulations, and experiments. IEEE Transactions on Robotics 24 (1), 88–104.

Transeth, A. A., Pettersen, K. Y., Liljebäck, P., 2009. A survey on snake robot modeling and locomotion. Robotica 27 (07), 999–1015.

Wang, X., Li, M., Wang, P., Guo, W., Sun, L., 2012. Bio-inspired controller for a robot cheetah with a neural mechanism controlling leg muscles. Journal of Bionic Engineering.
URL http://www.sciencedirect.com/science/article/pii/S1672652911601200

Yagnik, D., Ren, J., Liscano, R., 2010. Motion planning for multi-link robots using artificial potential fields and modified simulated annealing. In: Mechatronics and Embedded Systems and Applications (MESA), 2010 IEEE/ASME International Conference on. IEEE, pp. 421–427.

Ye, C., Hu, D., Ma, S., Li, H., 2010. Motion planning of a snake-like robot based on artificial potential method. In: Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on. IEEE, pp. 1496–1501.

Appendix

Videos of simulations and experiments

The following demonstration videos are included in the digital attachment for the Master's Thesis, but can also be found by following the given web links.

Spring mode control

- SpringModeControl.mp4
- https://goo.gl/yLVKmR

Propulsion method

- PropulsionDemo.mp4
- https://goo.gl/AlWL6S

Communication

- CommunicationDemo.m4v
- https://goo.gl/wkDN0I