NTNU – Trondheim
Norwegian University of
Science and Technology

Design, Testing and Validation of the ChIRP Robot Platform

Christian Berg Skjetne

Master of Science in Informatics
Submission date: April 2015
Supervisor: Anders Kofod-Petersen, IDI
Co-supervisor: Jo Skjermo, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Christian Berg Skjetne

Design, Testing and Validation of the ChIRP Robot Platform

Master Project, Spring 2015

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

In this thesis we will explain the design process of the ChIRP robot platform. We will also test and validate the platform as a tool in cooperative AI research. The testing and validation will be conducted as three separate experiments where each experiment will demonstrate some aspect of the platform. The contributions of this thesis is the development and testing of a new robotic platform for use in cooperative robotic research. We also hope the experiments will serve as an inspiration to future work on the robot platform.

Preface

The ChIRP robot platform was designed and built by the ChIRP group; Christian Berg Skjetne, Robert Versvik, Anders Søbstad Rye and Håvard Schei under the supervision of Prof. Pauline C. Haddow, and by Christian Berg Skjetne (Scientific Assistant) at the Norwegian University of Science and Technology.

The experiments discussed in this thesis have been designed and conducted by Christian Berg Skjetne in conjunction with the ChIRP robot group at the Norwegian University of Science and Technology in 2013 and 2014. The work has been supervised by Prof. Pauline C. Haddow, Prof. Anders Kofod-Petersen and Adj. Ass. Prof. Jo Skjermo.

Special thanks to the ChIRP group, Robert Versvik, Anders Søbstad Rye and Håvard Schei and to Edouard Bertin, Erik Samuelsson, Magnus Ulstein, Filip Fossum, Andreas Hagen, Hong-Dang Lam and Simon Glisic Randby especially for their help with setting up and running the experiments. Thanks also to Dr. Jean-Marc Montanier for lots of help and many interesting conversations.

Christian Berg Skjetne
Trondheim, April 29, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter covers the motivation behind the thesis and the goals we are trying to achieve. The last section in the chapter will give an overview of the structure of the rest of the thesis.

1.1 Motivation

The motivation behind the Cheap Interchangeable Robot Platform (ChIRP) project came from our own experiences doing research on cooperative AI systems using more established platforms. This work prompted us to create our own platform in an attempt to address the issues we had. The main limitations we were faced with were the price and expandability of existing platforms.

To address the price of the robot we created an inherently expendable robot that consists of a basic main robot with a less powerful microcontroller and just the basic sensors. If an experiment requires more processing power or additional sensors, both can be added to the robot as needed. By relying on expansions to add features as needed the cost and complexity of the robot can be kept at a minimum.

With the wide variety of different experiments being conducted in the field of robotics, the ability to tailor the robot platforms abilities to suit a specific experimental setup is often a requirement. In many cases the experiments require some special type of technology to be available on the robot such as a specific type of sensor or some form of communication system. In these cases the researcher may either be forced to choose between the few platforms that have the specific required technology, or she may choose a platform that can be extended with additional features either by using add-on modules or by creating a custom module tailored to the specific experiment. Some robot platforms address this problem by adding a lot of features to the robot in an attempt to cover as many possible experimental setups as possible. This however has the adverse affect of unnecessarily driving up cost and complexity of the robot as most experiments will only require a subset of the features available. The solution we have chosen is to make a simple robot that has a minimum set of features required in most experiments, and allowing

any further features required to be added as expansion modules. This enables the robot to be tailored specifically to any given experimental setup, keeping cost and complexity to a minimum.

The cost of investing in a new robot platform may make any researcher apprehensive about technology that has not been proven to work in research. Furthermore, while the price may be an important factor in choosing a platform, testing and development time may in many cases be of an even greater importance. A robot may be cheap to buy, but time is arguably the most scarce resource in many research projects. A researcher may not want to waste time on testing out a new platform when there are alternatives available that have a proven track record. This is why we have conducted several proof of concept experiments to test and validate the platform.

If we want our robot to help contribute to the field of robotic research, we will first have to prove that the robot can enable the researcher to make a contribution to the field. For the reasons stated above it may be unrealistic to expect researchers to start using the new platform without us first doing experiments our selves to prove that the platform can meet the requirement of the community. Other robot teams like the e-puck [Mondada et al., 2009] and the r-one [McLurkin et al., 2013] have done similar validation and testing work with experiments conducted by the development team and others. Both teams have also tested the robot as an educational tool for student. The ChIRP have been used in a similar capacity as a tool for helping younger students learn geometry [Stølsvik and Utgaard, 2014].

The motivation behind this thesis can be thus be divided in to two: first, we want to build an affordable modular robotic platform that can be used in robotic research with a special emphasis on cooperative systems; secondly, we want to conduct some proof of concept experiments to test and validate the robot platform. While this is our main motivation, we also hope this thesis can serve as inspiration as to how researchers may implement certain aspects of their own experiments using the ChIRP platform.

The motivations that are specific to each of the separate experiments covered by this thesis will be discussed in Chapter 4.

1.2 Goals and Research Questions

The design and construction of the robot, and the experiments discussed in this thesis them selves are separate in the goals they are trying to achieve. However, the goal of the thesis as a whole is to build and evaluate the ChIRP robot platform and draw a conclusion on the viability of conducting robotic research on the ChIRP robot platform, based on the collective success of the experiments.

Goal 1 *Design and build an affordable modular robot.*

Goal 2 *Test and validate the abilities of the robot platform in a cooperative AI research environment.*

Based on these goals, we have highlighted three research questions that we want to answer in this thesis.

Research question 1 *Can we design and build a robot fulfilling all of the design requirements?*

The design and construction of the robot is guided by a set of requirements we have identified while working with cooperative robotic research. In addition to these requirements, we have added features that could be improved upon on other robot platforms. The requirements can be seen in Table 1.1. A list of other available robotic platforms is available in Appendix A in Chapter 5.3.

Requirement	Description
Small size	Enabling experiments in smaller labs with many robots.
Affordable	Low price compared to other commercial platforms
Open source/HW	Enabling custom modules or changes to be made
Expandability	Enabling the robot to be tailored to an experiment
Long battery life	4+ hours of battery life for long running experiments

Table 1.1: Design requirements

The experiments conducted in this thesis are split in to two test cases and one validation case. The test cases will be used to confirm the usability of the platforms hardware, and to test different ways of using sensors and communication hardware.

The goal of the validation experiment is to conduct a larger scale AI experiment using neural networks and genetic algorithms and also to test the limits of the robots processor in terms of computing power and memory when used in this context.

Research question 2 *Are the ChIRP robots sensors and actuators accurate and precise enough to be used in cooperative robot experiments?*

To be able to conduct any robot experiment the researcher is at the mercy of the sensors, actuators and communication equipment available. The test experiments are conducted to confirm that the ChIRPs hardware is reliable and precise in an actual experimental environment.

Research question 3 *Is the microcontroller used in the robot powerful enough to be used in representative cooperative AI research?*

The validation experiments goal is to use the lessons learned during the tests to perform an actual experiment using AI methods.

1.3 Research Method

When designing the robot we start by defining what type of research the platform will target. Having defined this we then make a list of requirements for the robot. When the requirements are done the design process starts. First we find and test different components such as motors, controllers and sensors. When we have

identified the components we wanted to use, we design the circuit boards and create a prototype. The prototype include all of the components and gives us a crude robot that can be used to confirm that we have met all of the requirements. When we have gathered the data and confirmed that the robot behaves as expected, we design a chassis and wheels and order the circuit boards to be made by a professional manufacturer.

To confirm that the finished robot functions as expected we design and run three experiments and record the outcomes.

The first two test experiments function as proof of concept demonstrations of the capabilities of the platform as an educational and research tool. Both of these experiments are also deployed as actual demonstrations during high traffic functions at the university and at conferences. The nature of these functions makes it hard to control the surrounding environment in terms of light, sound and vibrations. The constant flow of participants, children as well as adults, wanting to get a closer look at the robots also means that we get to test the durability of the robots in terms of handling.

The last validation experiment is designed and run on a simulator developed for the ChIRP robot. The results of this experiment is then deployed on the actual robot for confirmation.

1.4 Contributions

The main advantages to the robotic platform are shaped by the design requirements that guide the development of the platform stated in Table 1.1 and can be summed up as the following:

- *Small size;*
- *Low price compared to other commercial platforms;*
- *Open source, open hardware;*
- *Expandability and*
- *Long battery life.*

A more in depth discussion of the design requirements can be found in Chapter 2 Section 2.1.1.

The other contributions of this thesis is the confirmation of the platform as a viable robotic research tool. We also aim to show that the hardware and software is stable and easy to use in this context. The expandable nature of the platform is a key feature that we will explore using different communication and sensor modules and some applications of these. Finally we hope that application of these modules can serve as an inspiration for future researchers using the platform.

The ChIRP robot have been featured in several functions with the aim of getting more kids and adults interested in technology and AI. Some of theses functions include:

- *Researchers Night (Norwegian University of Science and Technology. 2013, 2014);*
- *Teknologidagene (Hosted by the Norwegian Road Administration in 2014);*
- *Featured in TV segment on platooning (TV2, God Morgen Norge, 07.10.2014);*
- *Trondheim MakerFaire 2014;*
- *2nd International Conference on Robot Intelligence Technology and Applications.*
- *Featured in TV segment on Artificial Intelligence (NRK, Schrödingers katt 16.04.2015)*

It has also been used several times as a demonstration for visiting high school students at the Norwegian University of Science and Technology in 2013 and 2014.

1.5 Thesis Structure

This thesis contains three separate experiments that test different aspects of the ChIRP robot platform, but the over arching motivation for all of the experiments is to test and validate the viability of doing actual research on the platform. With this in mind we will treat all of the experiments as a single testing and validation case and the reflections and discussions given will be relevant to all of the experiments unless a specific experiment is mentioned. Only in the chapters describing the experimental architecture and model (Chapter 4) will we cover in depth the separate experiments one by one and the motivations that specifically pertains to a given experiment.

The introduction chapter (Chapter 1) will give a general overview of the motivation and goals for the thesis. Chapter 2 will give a brief overview of the robot platform for readers that are unfamiliar with it. Chapter 3 will give the reader the necessary background theory required to understand the concepts covered in later chapters. Chapter 4 will go in to each of the separate experiments in detail and explain the motivation, model and results for each of them. The last chapter (chapter 5) will evaluate the experimental results with regard to the goal of the thesis. The last chapter also covers the contributions and future work.

Chapter 2

The ChIRP Robot Platform

In this chapter we will give a short introduction to the ChIRP (Cheap Interchangeable Robot Platform) robot platform. Some of this work has been published in *Advances in Intelligent Systems and Computing volume 274* Skjetne et al. [2014].

2.1 Design and Testing Process

This section will give an overview of the method used when designing the ChIRP robot.

2.1.1 Design Requirements

In the first stage of the design process we identified the key requirements for the robot platform. The requirements were selected by looking at the different features implemented by other robot designs, and selecting the common minimum features we deemed necessary for most cooperative AI systems. We also selected features that we found lacking in some of the other platforms. The following list shows the selected requirements with an explanation of each point. A list of other robot platforms is available in Appendix A in section 5.4..

- *Small size* - If we want to create a comprehensive cooperative AI system, a large number of agents may be required. If the robot is too large the experiment will require a large area of operation. Many researchers do not have such facilities available to them. Larger robots also requires more space for storage and transportation. Finally, larger robots are also usually more expensive since they require larger motors and batteries.

- *Affordable* - The cost of acquiring a large number of robots may be a large obstacle for anyone wanting to perform experiments on physical robots.

- *Open source hardware and software* - It is impossible for any manufacturer of robots to predict the type of experiments that researchers want to conduct

on their platform, so we believe that all of the aspects of the design should be available for modification by the user. Another upshot of this model is that good hardware and software modifications may be shared freely and adopted by others in much the same way research it self is an iterative process of adaptation and modification of ideas.

- *Expandability* - Since we can not predict all of the hardware requirements for all experiments that researchers want to conduct, the only logical step if we want the price to be low is to enable specific hardware to be added when needed. This requirement also works great with the open source model. If a researcher requires a specific sensor or actuator she is free to design and add this feature at any time. The extension may then be freely adopted by other researchers that have similar requirements at a later date.

- *Long battery life* - Some experiments are required to run for a long time. On many of the robot platforms we tested the battery life was a limiting factor for the type of experiments that could be run on them. By using standard Lithium Ion battery cells that can be charged without removing them from the robot. This enable the batteries to be easily replaced when they are nearing the end of their life cycle. The batteries can also be charged while the robot is online with the running program in a sleep state, effectively making the theoretical maximum running time of an experiment infinite.

2.1.2 Testing and Prototyping

Once we had identified the requirements for the robot, we started the design phase. We identified distance sensors as the most fundamental of sensors for research based on experience and other robot designs. We also identified differential wheeled drive system as the best suited as it enables the lowest turning radius and the smallest footprint. When the basic sensor and actuator were selected we started testing different hardware from different suppliers to achieve the best results for the lowest price. The motors had to be comparable in speed and precision to other robots on the market, while still maintaining a low price. The selection of distance sensors was harder as we not only wanted them to be comparable with the sensors available on other robots, but to exceed them in terms of maximum sensing distance and noise sensitivity from ambient light.

A few motors were implemented and tested before we landed on the 28BYJ-48 stepper motor as the most affordable motor with the correct speed, precision and voltage levels for the robot. The distance sensors went through a larger set of tests including maximum sensing distance, field of view and sensitivity to ambient light. After testing several sensors from different manufacturers we selected the TSAL 6100 IR emitter paired with the BPW 41 N IR photo diode as the best suited. Data from the test of the selected sensor can be seen in figure 2.1. The angular response test was conducted with the motors running at top speed simulating actual noise levels. The distance test was conducted with the motors off and the fluorescent lights in the lab on.

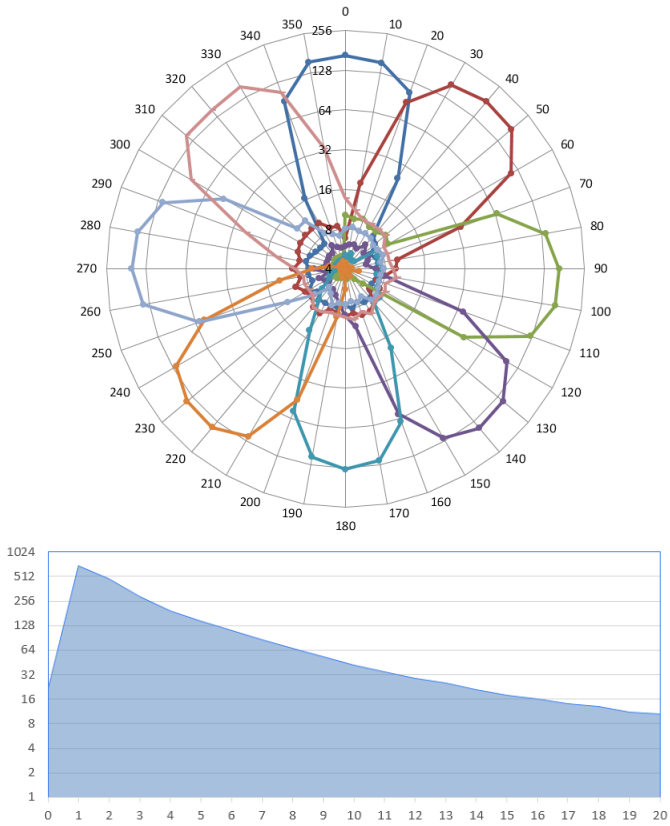


Figure 2.1: Test data for infrared distance sensors. The results are on a logarithmic scale. Top: Angular response test. The sensing target was placed 35-40mm from the robot and rotated around the robot in steps of 10 degrees. Bottom: Distance response test. The sensing target was moved towards the robot at predetermined intervals between 0 and 20cm.

When choosing the battery for the robot, the most important factors were availability and battery life. We chose to use LiPO batteries because of their high energy density and because they are readily available from multiple suppliers. We chose to use a 2600mAh single cell battery for testing the battery life, but larger or smaller capacity cells may be chosen base on price and application. The tests were conducted with the robot running both motors at full speed and with constant polling of the distance sensors. This yielded a battery life of about 4 hour.

Once the hardware had been tested and selected we could design and build the circuit boards and the chassis and build the finished prototype. A picture of the prototype can be seen in figure 2.2.

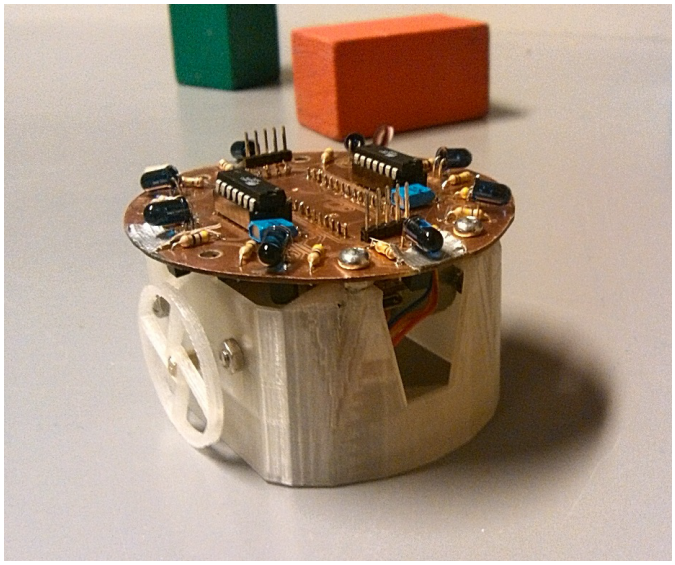


Figure 2.2: The finished ChIRP prototype.

2.2 Hardware Design

The ChIRP robot consists of a cylindrical chassis with two wheels attached to the side and a printed circuit board on top. The robot without any extensions can be seen in Figure 2.3 where *A* indicates the ATtiny84 coprocessors responsible for the 8 infrared sensors (four each) indicated by *B*. *C* and *D* indicates mounting holes for physical extensions like a camera or gripper arm and pin break out for adding electronic hardware extensions respectively. Not shown in the figure is the motor driver hardware, the motors and the main controller board situated inside the chassis. Table 2.1 shows some of the technical hardware specifications for the robot.

To enable the robots main microcontroller to run code uninterrupted and also to free up the I/O pins to enable extensions to be added, the sensor array and motors are controlled by three additional smaller microcontrollers. Two of these microcontrollers are attached to the sensor array and the last control the two stepper motors. These microcontrollers communicate over an I^2C bus that can also be used used to expand the capabilities of the robot with additional hardware. A functional diagram of the robot is shown in Figure 2.4. In the diagram we can see how all of the components are connected together. In the left column we have the main processor that is running the user program. When the user program wants to read a sensor or update the speed of the wheels, a command is sent over the I^2C buss to the coprocessors (shown in the third column marked ATtiny84). The coprocessors keep track of all of the sensors and actuators independently. If the user program wants to read the distance sensors the sensor controllers simply

reply with the most current measurement in its memory back over the I^2C buss. If the user program wants to update the motor speeds, the motor controller just takes in the new speeds and updates the internal memory before it returns to its main job of stepping the motors.

Full assembly instructions for the ChIRP robot is available in Appendix B in section 5.5.

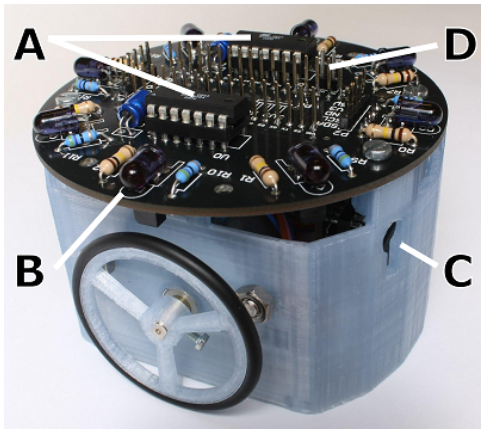


Figure 2.3: (A) Sensor coprocessors (B) IR sensors (C) Mounting holes for physical hardware (D) Pin break out for electronic hardware

Size (mm) HxW	55 x 85
Actuators / Sensors	Differential wheel drive / 8 IR distance sensors
Battery mAh (hours)	2600mAh (4 hours)
Main Microcontroller	Atmel ATmega32u4
Secondary microcontrollers	3x Atmel ATtiny84

Table 2.1: Hardware specification

2.3 Software

The robot programming environment is based on the Arduino IDE[1]. Specifically the robot uses the Arduino Micro development board. The choice to base the robot on the Arduino environment was motivated mainly by two things: firstly, the Arduino environment adds an abstraction layer on top of the AVR micro controller that simplifies hardware interaction allowing programmers with limited experience

[1] Arduino homepage: http://arduino.cc

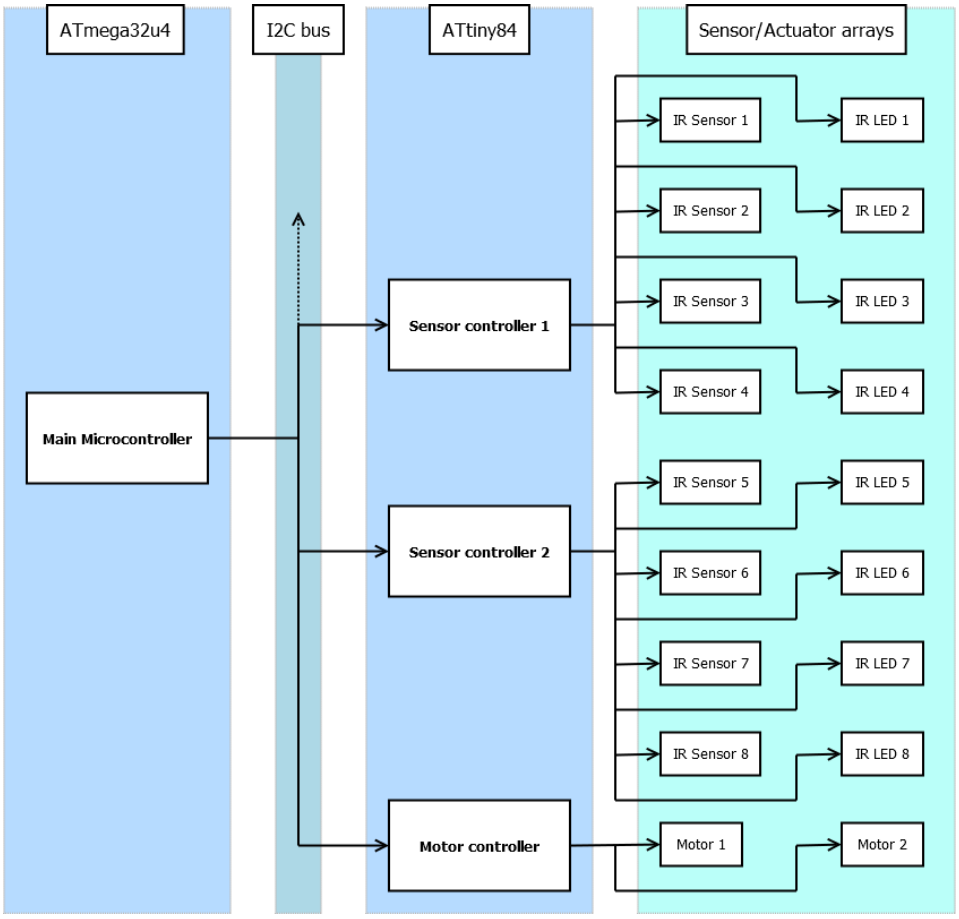


Figure 2.4: Functional diagram of the ChIRP robot

with programming on micro controllers to quickly start writing code. The abstraction layer is implemented on the IDE side allowing more advanced users to use the default programming environment for the AVR micro controller side by side with the simplified Arduino environment. The second reason is that due to the extensive user base of the Arduino a large amount of third party libraries are available potentially reducing development time.

To help users of the robot to quickly get in to programming on the robot, we have created an extensive set of libraries that are available for download. These libraries simplifies interaction with Motors and sensors as well as some of the extensions. Listing 2.1 shows a very simple object avoidance program using the distance sensing and motor libraries.

Since the robot utilizes more than one micro controller the code that runs

on the sensor and motor coprocessors are also provided. Reprogramming these can be done using the Arduino programming environment, but requires additional programming hardware.

All of the software for the ChIRP robot is open source and available for download from the ChIRP webpage[2].

```
//
//    Simple object avoidance example program
//

#include <Wire.h>
#include <IrDistCom.h>
#include <Motors.h>

int threshold = 50; // Threshold for object detection

// Initialize the distance sensor and motor objects
IrDistCom ir;
Motors motors;

// This array holds the distance measurements
unsigned short sensorData[8];

// This method is run once when the robot first boots
void setup()
{
  // Join i2c bus
  Wire.begin();
}

// This method runs continuously as long as the robot is turned on
void loop()
{
  // Update the distance measurements
  ir.getDistSensors(sensorData);

  // If an obstacle is detected, turn away from it.
  // If not keep going forward.
  if(sensorData[7] > threshold && sensorData[0] > threshold)
  {
    motors.moveAtSpeeds(500,-500);
  }
  else if(sensorData[1] > threshold || sensorData[0] > threshold)
  {
    motors.moveAtSpeeds(-500,500);
  }
  else if(sensorData[7] > threshold)
  {
    motors.moveAtSpeeds(500,-500);
  }
  else
  {
```

[2]ChIRP homepage: http://chirp.idi.ntnu.no

```
47      motors.moveAtSpeeds(500,500);
48    }
49
50    // Wait 200 milliseconds before re-evaluating
51    delay(200);
52  }
```

Listing 2.1: Example of a very simple object avoidance program

Chapter 3

Background Theory

In this chapter we will explain the concepts and theories that forms the basis for the experiments. This section is not divided in to separate sections for each of the experiments since some of the concepts covered are relevant to more than one experiment. The first part of this chapter explains the concepts behind the AI methods used in our experiments while the last two sections introduce image segmentation in HSV colour space and platooning as general concepts used in the experiments that may be useful to know about.

3.1 Subsumtion Architecture

Subsumtion architecture is a useful way to organize a robots behaviours in a priority hierarchy based on how important the behaviour is to the success of the robot.

Subsumtion architecture was developed by Rodney Brooks and described in his seminal paper [Brooks, 1986] on layered control systems for robots in 1986. The architecture is based on the end goal of a robotic controller being divided in to sub goals. The behaviours needed to achieve these sub goals are then ordered in a hierarchy based on their priority.

Subsumtion can be seen as an alternative to the more traditional sense-plan-act model of solving a task where the sensor input is fed in to the controller which then makes a plan to achieve the target goal and at last acts on the decided plan. In a subsumtion model the input from sensors are fed in to the different behaviour layers in parallel and each layer determines what actions, if any, are required to achieve the given layers goal. The priority hierarchy of these layers are then used to decide what actions to take.

We can think of the subsumtion architecture as being similar in some way to Maslow's hierarchy of needs [Maslow, 1943]. In Maslow's theory the needs of humans can be divided in to sub-goals and placed in a hierarchy where the behaviour needed for success of the lower, more fundamental needs supersedes the secondary, higher needs. In other words, the need for food, water, shelter and safety supersedes the human need for friendship and love, the need for self-esteem and

respect and the need for self-actualization. This is similar to the way for instance a robot vacuum cleaner can use subsumtion to prioritize things like not crashing in to walls and avoiding getting stuck, exploring and mapping the environment and vacuuming up dust.

3.2 Swarm Robotics

Swarm robotics is an approach to robot coordination that takes inspiration from the way social insects and animals like ants, bees, birds and fish cooperate to complete tasks that would be hard or impossible to do for a single individual. [Garnier et al., 2007]

As with many of the terms used to describe domain specific systems, the swarm robotics term is subject to ambiguity and a clear definition of how the term is understood and used by a researcher is required. Şahin [2005] defines swarm robotics as *"... the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment."*

This definition, while helpful to understand in simple terms what we mean when we say swarm robotics, is not sufficiently restrictive when trying to classify research. In addition to this definition Şahin suggests some simple criteria to help narrow down the term. The first of these criteria is that the robots should act autonomous. This requirement entails that a robot has to have physical embodiment with the ability to interact with the world on its own. The next criterion is that we should have a large number of robots. What size of swarm the word "large" covers is hard to clearly define, but at the very least a scalability in group size should be the aim in a swarm robotics system. Next we have the homogeneity of the agents in the swarm. The system studied should consist of relatively few homogeneous groups of larger size. The abilities of one individual in the swarm should be limited and solving a given task should be hard or impossible for a single robot to do on its own. While a task could be solvable for a single robot, at the very least a significant improvement in robustness and/or performance should be achieved when more robots cooperate to solve the task. Finally we have the requirement of local sensing and communication. The robots in the swarm should only have the ability to sense and communicate over short and limited distances. This requirement ensures that the coordination of the group is distributed over the population.

It is important to note that while we refer to these criteria as requirements, any given study that does not meet one or more of the requirements may still fall under swarm robotics. The adherence to the criteria is only a measure of the degree to which the term swarm robotics might apply.

3.3 Artificial Neural Networks

Artificial Neural Networks is a common controller schema for robotic application and was used in our experiments to examine the capabilities of the ChIRP micro-

controller.

The history of neural networks can be traced back to the very beginning of the field later to be known as computer science. The field of computer science was started in the 1930's and '40s with the first formal definitions of computability. While the von Neumann architecture of computing emerged as the winner in the end, the viability of the other models has never been dismissed and are still studied today. One of the competing models was the neural network. This computational model is inspired by biology and the physiology of neurons. These models do not operate sequentially in the way Touring machines do.

In a conventional von Neumann architecture the minimal set of functions required for universal computation is defined by the processors instruction set. In a neural network, these primitive functions are the result of the nodes and the network topology itself. A neural network can be seen as a network of primitive functions where values are transmitted between nodes via edges. The first computational model based on real neurons was described by Warren McCulloch and Walter Pitts in 1943 [McCulloch and Pitts, 1943]. In this model the neurons are defined as either outputting a signal or not based on whether the combined input to the neuron exceeds a set threshold or not. By allowing both inhibitory and excitatory inputs to a neuron, where the neuron can be inactivated by a single inhibitory signal, we can express all logical function with a McCulloch-Pitts network [Rojas, 1996]. By allowing weights to be used on the edges of the network we can simplify the topology of the network. Figure 3.1 shows a simple representation of an abstract neuron with n weighted inputs. First the input values x_n are multiplied with their corresponding weights w_n then added together. Finally the primitive function f is applied.

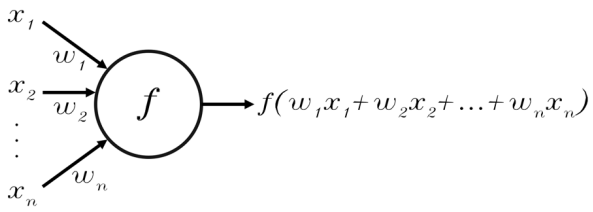


Figure 3.1: An abstract neuron with n inputs (adapted from Rojas [1996])

While any weighted network can be shown to be equivalent to a McCulloch-Pitts network, the ability of weighted networks to simplify the network topology gives the weighted network the ability to represent a larger array of functions by simply altering the variables instead of altering the topology itself. This ability to alter the network function using variables simplifies the process of learning. In 1958

Frank Rosenblatt proposed the *Perceptron* as an alternative to the McCulloch-Pitts model. The perceptron introduced numerical weights and a special interconnection pattern. Rosenblatt also introduced the *Perceptron learning algorithm* that works by adapting the weights of the of the network to reduce output error. Later the model was refined and its computational properties studied in depth by Minsky and Papert [1969]. Although the previously mentioned authors are among the most influential in the development of the first artificial neural network models, several other scientists have made great contributions to the field both in these early years, all the way up till today. This has resulted in a great variety of ANN models available to researchers today. For the sake of brevity we will not discuss all of the different models but rather name some of the more popular properties of networks used in research today.

One of the most common topologies used in ANNs is the *feed-forward network* [McCulloch and Pitts, 1943]. This is the type of network we have been focusing on in this section. We can define a feed-forward network as a graph with nodes able to evaluate a single primitive function and directed edges able to transmit the numerical results from node to node.

To enable an ANN network to implement a complex network function a large, multi-layer network may be required. To help with determining the values for such a large amount of weights a learning algorithm is used. The one of the most common learning algorithms used today is the *Back-propagation algorithm* [Rumelhart et al., 1988]. This works by minimizing the error function in weight space by gradient decent. This in turn requires the unit function to be continuous and differential and thus excludes the traditional threshold function since we have to calculate the gradient of the error function at each step. One of the most commonly used functions is the *Sigmoid* function shown in equation 3.1. By increasing the constant c the slope of the function increases and the function becomes a closer approximation of the step function. A plot of the function for $c = 1$ is shown in figure 3.2.

$$f_c(x) = \frac{1}{1 + e^{-cx}} \tag{3.1}$$

The *back-propagation algorithm* is not used in the experiments for reasons discussed in later chapters, and the inner workings of the algorithm are therefore outside the scope of this thesis. Instead the experiments use evolutionary methods to determine the weights of the neural network.

3.4 Evolutionary Algorithms

Evolutionary algorithms is common way to find parameters for different robot controllers. While it is possible to run an evolutionary algorithm on robots in real time it is more common to run the algorithm in a simulator to speed up the process. In our experiment we will do the latter.

At the end of the 18th and start of the 19th century naturalists started to scientifically approach the problem of diversity in nature. While artificial breeding

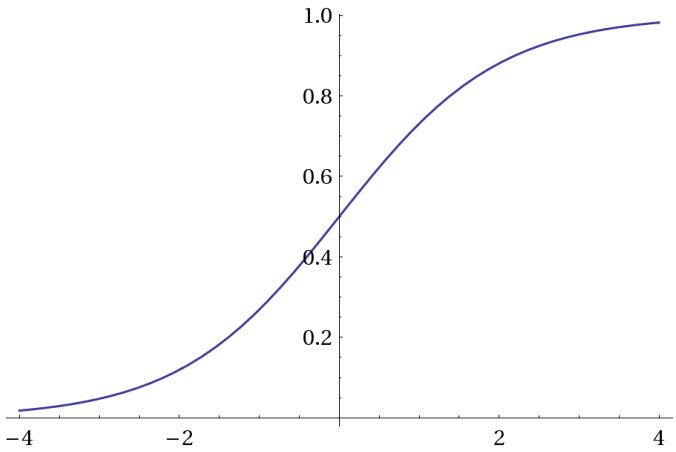


Figure 3.2: The sigmoid function for $c = 1$.

of animals to enhance favourable and suppress unfavourable traits had been used by humans both consciously and subconsciously for millennia, it was only then scientists began to formulate theories as to how similar mechanics could be working to shape the living world at large. While the most famous book on the subject of natural selection is Darwins ground breaking *"On the origin of species"* [Darwin, 1859], Darwins ideas were built upon the work of both contemporary scientists and much older ideas. Indeed even some of the ancient philosophers wondered about how nature and the struggle for survival could be shaping the diversity seen in nature [Aristotle, Physics lib.2, cap.8]. These and similar ideas re-emerged in the 18th and 19th century with the works of people such as Edward Blyth and Thomas Malthus to name but two. Both Blyth and Maltus' ideas served as inspiration for Darwin when he put together his theory on natural selection. Darwin defined his theory on natural selection as the *"principle by which each slight variation, if useful, is preserved"* [Darwin, 1859].

In the formative years of the electronic computer in the 1950s and 60s, several scientists started developing evolution-inspired algorithms for optimization and machine learning. A lot of this early work have been largely overshadowed by the attention that evolutionary strategies, evolutionary programming and genetic algorithms have received [Mitchell, 1998]. *Genetic algorithms* (GAs) were pioneered by John Holland with his book *Adaptation in Natural and Artificial Systems* [Holland, 1975]. Together with his students and colleagues at the University of Michigan, he started working on GAs in the 1960s and the 1970s. In contrast to other researchers that attempted to use ideas from evolution to solve a specific problem, Hollands original goal was to study in more general terms how adaptation occurs in nature and how it could be adapted for use in computer systems [Mitchell, 1998]. With this in mind, Holland introduced a framework closely mimicking many of the processes found in nature. The general tenets first formulated by Holland are

still descriptive of many of the GA systems implemented today. Most GAs use variations of the same general building blocks:

- *Genotype* - This is the basic data structure representing the individual on a fundamental level. The genome has to be coded in such a way as to enable the extraction of features or traits. In addition it has to be constructed in such a way that a slight random alteration or *mutation* of the genome only results in slightly altered traits. Lastly, the algorithm has to be able to recombine two separate parts of two different parent genomes in order to create a new genotype that shares traits with both parents. This mechanism is called *crossover*.

- *Phenotype* - In most cases the genotype, given the rules outlined in the previous point has to be translated in order for it to create the behaviours or traits needed for evaluation of the individuals fitness. These traits, that the genotype encodes is called the phenotype. Usually the phenotype have to be tailored to the specific problem or task we wish to solve.

- *Fitness function* - These are the "rules" we use to determine to what degree an individual is capable of solving the given problem. These rules define the shape of the solution space that we wish to search. The formulation of a good fitness function is arguably the most important, and often the hardest part of creating a GA system. Since the fitness function defines the problem we wish to solve it generally has to be created specifically for that particular task.

- *Selection mechanism* - After the fitness of an individual has been determined using the fitness function we have to determine the scope of our search in the solution space. The selection mechanisms determine the probability of a given individual reproducing and being used to create a new individual based on the fitness calculated by the fitness function. By tweaking the selection mechanism we can change the balance of the search from exploration (larger coverage of the solution space) to exploitation (optimizing a few good solution to improve them even further.) The selection mechanism can also change over time to enable the best balance between these two. The only minimal requirement strictly needed to enable evolution is that there has to be, on average, a higher probability for an individual with higher fitness to create offspring that that of an individual with a lower fitness.

Using the building blocks described above we can make an outline of a general GA. Most GA systems follow a similar formula to the one stipulated below:

1. Start by creating a population of individuals with random genotypes.

2. Translate these genotypes in to phenotypes and evaluate their fitness using the fitness function.

3. Based on the fitness of the individuals in the population, select a set of parents for reproduction such that on average individuals with higher fitness reproduce with a higher probability than the individuals with lower fitness.

4. Now use the individuals selected for reproduction to create a new population using recombination (combining genotypes from two (or more) individuals to create a new individual sharing traits with both parents) and mutation (random changes to the genome are introduced to encourage exploration of the solution space, avoid stagnation and enable escape from narrow local maxima).

5. Once the new population has been created the cycle can start over from step number 2.

The criterion for stopping the loop depends on the experiment. If there is a specific target fitness to reach as a goal, then the loop would break once the solution has been found. In most cases however, no such target fitness exist. To deal with this, the researcher has several options. The simplest method is to wait a fixed number of generations an take the best solution achieved at the point. Another method is to observe the standard deviation over the population genomes or fitness and end the loop when the population has converged beyond a threshold value.

3.5 Image segmentation in HSV colour space

The tracking of the absolute location of robots in an environment is useful both for robot controllers and for recording the behaviour of robots. Many different systems for tracking the position of robots exist. A very simple and cheap way to do it is to use a camera to track markers placed on the robots. While more advance systems can use shape recognition to track markers with patterns, a simpler approach is to just use different coloured markers and a threshold to differentiate the colours from the background. This section will explain what HSV is, and the reason for using the HSV colour space when doing image segmentation.

In colour based image segmentation one tries to separate out a region of colour space that corresponds with the colour in a given region of interest (ROI) in an image. While this may sound trivial for images with known or fixed lighting conditions, the problem becomes far harder to solve for more dynamic scenes. The classical RGB (red, green and blue) colour space is ill fitted for segmentation under these conditions. The problem the RGB colour space faces under these conditions is that the hue, saturation or chroma and colour value or brightness for a given point in an image are distributed between all three axes of the colour space making it hard to define simple threshold constraints for a given coloured object. The more sensible approach used in the HSV (Hue, Saturation, Value) colour space is to set the hue by it self as one of the axes, and the value and saturation on the other two axes respectively. In the RGB colour space red, green and blue can be found along their respective axes, and to get different shades (mixes) of these you have to also move along two or more of the axes. In HSV on the other hand, red, green and blue can all be found along the rotational hue axes from 0 and 360 degrees for red to 120 and 240 degrees for green and blue respectively. This allows us to define simple threshold constraints for a set of colour shades. A graphical representation of the HSV colour space can be seen in figure 3.3. The problem of boundaries

when dealing with the uncertainty of lighting conditions in images makes demands to the colour space similar to the philosophers in the The Hitchhiker's Guide to the Galaxy's demands in that it requires *'rigidly defined areas of doubt and uncertainty'* [Adams, 1979] something that is hard to do in the RGB colour space.

As for other colour space representations Zakir et al. [2010] have done a comparative study of colour segmentation using different colour space representations for application in road sign recognition, and came to the conclusion that the HSV representation yielded the best results for this application under variations in lighting and environmental conditions. While the conditions in a lab are certainly under far more control than the conditions on the road would be, we could still face similar problems with respect to time of day and light reflections albeit greatly reduced.

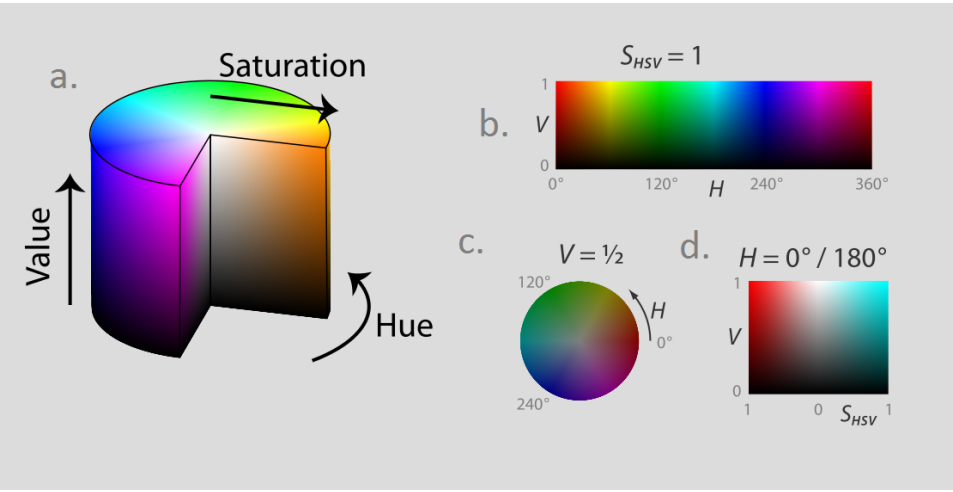


Figure 3.3: (a) cut-away model of the HSV colour space. (b-d) two dimensional plots of the model with one parameter at a constant value. (b) The cylindrical shell with a constant saturation of 1. (c) horizontal cross-section with a constant value of 0.5. (d) Rectangular vertical cross-section with a constant hue of 0/180° (Jacob Rus / Wikimedia Commons / CC-BY-SA-3.0 (adapted))

3.6 Platooning

Platooning is a good example of a cooperative robot task that may have uses in the automotive industry in the near future. It is also a useful task to test communication and position tracking of robots.

The term *platooning* or convoying refers to a system where a group of autonomous vehicles or robots navigate to and from the same location travelling in a line or formation where each agent is following the agent in front of it. The hope is that autonomous vehicles driving in platoons will be able to save on fuel

by reducing drag due to air resistance. Other potential benefits include lowering congestion, increasing road safety and the possibility for unattended driving for long haul transportation.

Variations on platooning tasks have been tested many times using both autonomous vehicles and smaller robots. Bergenheim et al. [2012] has written an overview of five current projects focusing on vehicle implementations. These systems vary in what sensors they use, infrastructure requirements, how they integrate with existing traffic and the specific goals they were optimized for. Between them, these experiments cover many of the of the goals and challenges platooning systems will have to face before we can expect to see such systems in commercial use. As an example of the numerous experiments conducted on smaller robots we can name the fuzzy logic based system of Marapane et al. [1994]. In contrast to many systems, this controller was designed to use a purely visual tracking system without the need for communication between robots. While communication hardware has become increasingly affordable and reliable over the last decade, it is still subject to external interference and any scheme that utilizes communication will still be required to operate safely if communication is interrupted.

Compared to many of the other self driving systems researchers experiment with today the possibility of this specialized form of autonomous driving is not greatly hindered by any technical obstacles. Most of the required sensors and hardware are available in some way in modern cars today. The reason for the delay in actual commercial systems may be traced to a more human desire to feel in control. While there are other obstacles like legislation, required system standardization across car manufactures and the lack of specialized infrastructure, the public will have to accept that a computer is better at driving a car than we as humans are before we can expect to see real life systems in use by the public. This change may be already happening, and in some sense it has been happening for several years.

For years now we have seen that lifesaving automatic systems have been introduced by a few car manufacturers to start with only to be implemented in almost every car, and in some cases required by law only a few years later. These systems range from a simple warning if seatbelts are not fastened to computer based systems like traction control and anti-locking brakes. More obvious comparisons can be made to modern systems that we are starting to see in production vehicles today like adaptive cruse control, automatic emergency braking and lane guidance systems. With these systems being able to take active control of the steering wheel, brakes and accelerator of the car plus the sensors they use to know the position and speed of the car and the cars around it, it is simple to see that the road to a self driving car is short indeed. With these systems starting to make their way in to more and more vehicles and their benefit starting to become clear one would expect at some point that the public will require them to be included in newer cars and thus become more comfortable with the idea of the car making choices and performing actions on their behalf.

Chapter 4

Experiment Architectures and Models

All of the experiments covered in this chapter have been run on the ChIRP robot platform. The two first experiments were used as demonstrations of the capabilities of the robot, while the final experiment was created to validate the robots abilities on a more complex AI problem using techniques and experiences developed under the previous two experiments.

4.1 Experiment I - Testing by human versus swarm game

This experiment was created as a proof of concept demonstration of the ambient light and bluetooth extensions. It was demonstrated at the Researchers Night event at the Norwegian University of Science and Technology in 2013 and 2014.

4.1.1 Motivation

The motivation behind this experiment was to test the ambient light sensors and the bluetooth hardware extensions. The experiment was also used as a demonstration for high school students visiting the university.

4.1.2 Experimental setup

The experiment was set up as a game where a human is controlling a robot attempting to cross an arena in the shortest possible time while a group of autonomous robots attempt to trap the player and slow her down. The human controlled robot being controlled over bluetooth from an app running on an android tablet is shown in Figure 4.1

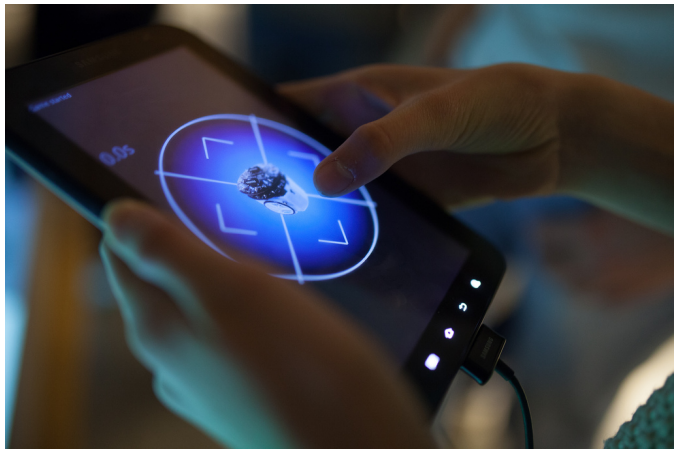


Figure 4.1: Robot being controlled over bluetooth from a tablet
Photo: Kai T. Dragland ©2013

The control of the robot is achieved via a two way bluetooth serial link between the robot and the tablet. In addition a projector is mounted over the arena projecting a playing field down on to the surface of the arena. The projected player field consists of a starting area, a moving "safe area", and an end area. The purpose of this is to allow the human controlled robot to know when it has left the starting area and the timer can start, when it is in the main game area and the other robots will attempt to stop it, when it is in the moving "safe zone" where the other robots can not see it and when it has reached the goal and the timer can stop. This is achieved using light sensors mounted on the robot that can sense the different light intensities of the projected image. An overhead representation of the game arena can be seen in figure 4.2. The game progress is as follows:

1. *Start* - The human controlled robot starts in the starting area situated in one of the corners of the arena.

2. *Game zone* - Once the robot is driven out of the start area a command is sent from the robot to the tablet starting a timer. The human driver now has to drive to the opposite end of the arena in the shortest possible time. When the robot is in the main game zone the infrared emitters on the robot turns on and the other robots that are in line of sight will start driving towards the robot and attempt to stop it.

3. *Safe zone* - If the driver wants to she may attempt to hide from the other pursuing robots by entering a moving "safe zone" present on the field. If the robot detects that it has entered the safe zone it will turn off its infrared emitters rendering it "invisible to the other robots. This may be a good strategy to adopt if your robot is being followed by many of the other robots

and you are having a hard time getting away. However, the safe area is always moving and keeping the robot inside the area for an extended period is hard. The timer will also keep running while you are in the "safe zone" so it is best used to confuse the other robots for a short time thereby getting a head start when you are ready to exit the zone.

4. *Goal area* - The goal area is located on the opposite corner of the arena from the starting area. When the driver gets the robot in to the goal area a command is sent from the robot to the tablet that the game is over and the timer can be stopped.

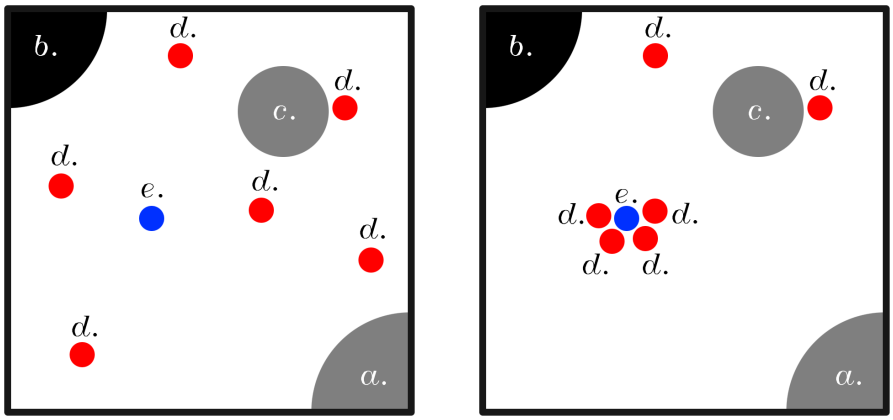


Figure 4.2: (Left) Overhead view of the game arena. (Right) Opponent robots attempting to stop the player robot. (a.) Starting area. (b.) goal area. (c.) "safe zone". (d.) opponent robots. (e.) player controlled robot.

This experiment can be seen as an extension of the box pushing experiment described by Skjetne et al. [2014] and Berg and Karud [2011], where several robots cooperate to push a box in the same direction. The same subsumtion architecture used in that experiment is utilized in this experiment, the only difference being that the goal here is not to push a box but to flock around a robot controlled by a human and stop it from crossing the arena. The subsumtion architecture behaviour is illustrated in Figure 4.3. For a more thorough description of subsumtion architecture see Section 3.1.

4.1.3 Results

The game worked according to our expectations. We were apprehensive about the lighting conditions at the demonstration venue interfering with the ambient light sensors so we added the possibility to adjust the threshold values on the fly using

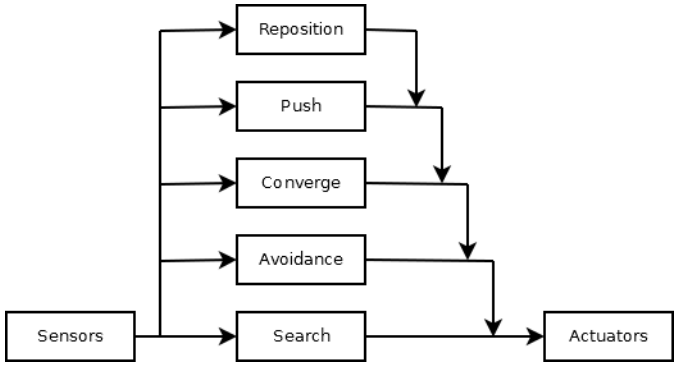


Figure 4.3: Box pushing behaviour (adapted from Berg and Karud [2011]).

the tablet simplifying the process. We ended up having to adjust the values only once after arriving at the venue. All in all the ambient light sensors worked better than expected. The possibility to use a projector to create a dynamic environment for the robot to interact with and sense opens up many possibilities for future experiments. The bluetooth extension also worked great. A very small number of players found the robot to be hard to control at first, but most players improved greatly after trying a second time.

4.2 Experiment II - Testing by platooning

This experiment was made as a proof of concept demonstration of the camera tracking and bluetooth control scheme. The experiment was used as a demonstration of platooning during the Teknologidagene conference in Trondheim, Norway in 2014.

4.2.1 Motivation

In this experiment we wanted to test using a camera to do positional tracking of the robots. We also tested using the bluetooth communication hardware from the first experiment to send and receive commands from a simulator running on a computer. We used the position data gathered from the camera to update the simulator and calculate the next movements for the robots. The commands were then sent over bluetooth and executed by the robots. Thus we had a control loop consisting of sense, plan and act that would function as a good test of the accuracy and latency of both the camera software and the bluetooth communication.

4.2.2 Experimental setup

The experiment consists of four ChIRP robots driving in a figure eight. By pressing a button the robots can toggle between platooning behaviour and behaviour similar

to human drivers. Platooning is described in more detail in Section 3.6. The experiment was not a simulation of any real world scenario but rather intended as a demonstration of the basic concept of platooning. A photo of the demonstration can be seen in Figure 4.4. In general the system works by going through the following steps in a loop:

1. A camera mounted over the driving area is used to detect the position and rotation of the robots using two markers placed on top of each robot.
2. The robot tracking software sends the position and rotation data to the simulation software which in turn updates the positions of all the robots and calculates the robot movements for the next time step.
3. The movement commands calculated by the simulation software is then sent to the individual robots over bluetooth.
4. Each robot executes the commands it has received and the process start again from step 1.

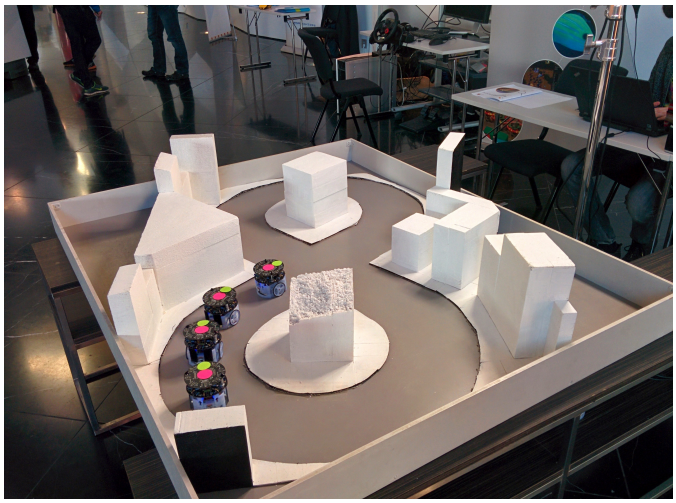


Figure 4.4: ChIRP robots demonstrating platooning.

The system uses the same bluetooth remote control feature that was implemented in the first experiment. It also uses a very simple subsumtion-like path planing algorithm. In addition to these features we also developed a way to track the position of the robots globally using a camera and markers placed on the robots. The tracking software works by following these simplified steps:

1. Images from a high definition camera mounted over the robot area looking down is recorded by the software.

2. The camera images are corrected for displacement with relation to the robot operating area by marking in the software the four corners of the area. Using these four points the image is corrected for variations in perspective and the x/y pixel-based coordinate system of the camera can be transformed in to a width/height fraction-based system. The fraction-based system is more suited for importing in to a simulation environment.

3. Next the software will first convert the RGB colour space of the camera image in to HSV before applying a series of thresholding functions to the image to separate out the two markers from other features in the image.

4. When a centre position marker feature is detected in the image the software checks to see if this is a robot it has detected before. If it has seen this robot before it will update the robot list with the new position, if it does not correspond to any of the known robots, the software will add it to the list and assign it a unique ID.

5. Next the software looks for rotational markers close to the known position of the robot. If found, the rotation, or heading of the robot is updated.

6. The last step of the process is for the software to broadcast the id, position and rotation of all of the detected robots over udp to be used by simulation software.

7. The process starts over from the top.

One problem we had to solve since we were using the same markers for all of the robots was how to distinguish which robot in the simulator corresponds to which bluetooth address. The solution we used was to create an assignment phase at the beginning of the experiment. We started by confirming that the camera was tracking all of the robots. Next we told one random robot over bluetooth to turn 90 degrees and stop. We then assigned the camera tracking id of the robot that had changed its angle the most to the bluetooth address we sent the command to. Next we sent a command to the robot to turn back to its original position. By performing the same few steps on all of the robots we now had all of the camera and bluetooth IDs assigned. This calibration step takes around three seconds for four robots to perform and only has to be done when the simulation is first started.

4.2.3 Results

The experiment was a success with all of the components working as expected. We were surprised to see that the latency of the system was quite low. The camera tracking worked great, not only in the lab, but also under the unpredictable lighting conditions of the conference hall used during the demonstration. This type of tracking could be a great way to model gps tracking or similar absolute positioning systems in future experiments. As we will see in the final experiment, it can also be used to simply gather positioning data from robotic experiments without giving feedback in to another system.

4.3 Experiment III - Validation by evolving ANNs

This experiment was created as a validation experiment run in our lab. In this experiment we evolved an artificial neural network to navigate in a road-like environment.

4.3.1 Motivation

The intentions of the first and second experiments was to test the hardware and software of the ChIRP robot platform. This third and final experiment was intended to demonstrate the viability of the platform by combining aspects from the two previous experiments in to a larger more complicated artificial intelligence system. This experiment also serves as validation for some hardware and software not covered by the other experiments.

We used genetic algorithms to create a neural network-based controller for the ChIRP robot. The controller was evolved in simulation and transferred over to the robot for validation. By doing this we can show that the simulator represents the physical robot in a good way. We can also demonstrate artificial neural networks running on the limited hardware of the ChIRP robot.

The inspiration for the experiment was Drchal et al. [2009] who used HYPERneat to create a neural network robot controller able to navigate a road network. The approach we have chosen is much simpler but it still yielded comparable results. However, the scalability of the HYPERneat approach is obviously not possible in our system.

A more general description on genetic algorithms and artificial neural networks can be found in Sections 3.4 and 3.3.

4.3.2 Experimental setup

The goal of the experiment was to create a controller able to navigate the road-like environment while introducing as little human bias as possible during the learning process. The reason we are not using the back-propagation algorithm is that it requires a priori knowledge about states in the system. To train the ANN with the back-propagation algorithm we require cases of known input states and corresponding output states to be able to calculate the error we need to propagate back through the network. These requirements makes the back-propagation algorithm unsuited for our experiment since we have no knowledge about the solution we want. In contrast, the only feedback a genetic algorithm needs is a measure of the degree to which a specific phenotype is able to solve the problem. This make us able to define simple rules of success and failure and thus keep the introduction of bias to a minimum.

The main building blocks on the software side of the system are described bellow.

- *Simulator* - We have made a simulator to enable fast development and test-

ing of robot controllers. The simulator is written in Java[1] and implements the Slick[2] wrapper for the LWJGL 2d graphics library[3] and the JBox2d physics engine[4]. The simulator supports running the physics engine at different speeds. This feature is critical when working on evolution where several hundred trials may be necessary before we can even begin to see desired behaviour. The simulation can also be paused and resumed during runtime. Full simulation in terms of the speed, acceleration, inertia and friction of the robots are simulated using the physics engine. The physics engine allows us to add physical objects, able to interact with robots and each other, to the simulator. The distance sensors and the ambient light sensors have been implemented and tested in the simulator.

- *Evoengine library* - To enable quick and easy development of Genetic Algorithms we have made a general Java library that enables the developer to implement a genetic algorithm with minimal effort. Several well known selection mechanisms have been implemented making it simple to test different methods. The implementation of phenotypes and non-standard genotypes are handled by simple interfaces. Once the programmer has implemented the interfaces, and set up the required parameters like mutation rate, crossover rate, population size etc. the library is ready to run the algorithm. Once started the GA will run in a separate thread allowing the rest of the program to deal with the results. The programmer may also choose to manually step the generations of the GA. This is required if the fitness of the phenotypes are dependent on external calculations done in a separate thread, as is the case with a simulator for instance.

- *ANN library* - We have also made a general Java library for developing artificial neural networks. The parameters and topology of the network can be set in a configuration file and the library will create the network automatically. Several popular connection schemes are supported and the library supports the back-propagation algorithm. To train the network using back-propagation you only have to feed inn training cases along with expected output and the algorithm does the rest. You can also set network weights manually and export network weights for later usage. This is useful when using the library with a genetic algorithm.

Once the basic building blocks for evolving an ANN-based controller were in place, we integrated the ANN and the GA in to the simulation environment. We constructed a road environment that can be seen in the simulator screenshot shown in figure 4.5

The road environment was constructed to have many turns and limited sections of straight road, since robots that are lucky enough to start on a straight road tend to skew the fitness function in their favour . This was observed in the first road

[1] `http://java.com`
[2] `http://slick.ninjacave.com/`
[3] `http://www.lwjgl.org/`
[4] `http://www.jbox2d.org/`

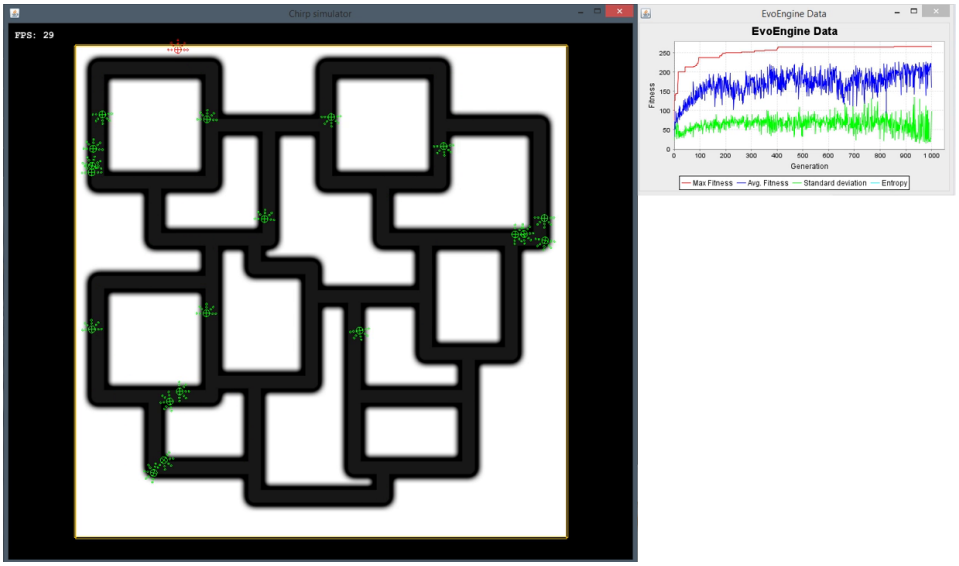


Figure 4.5: Screenshot of simulator.

environment we tested. This environment was constructed as two large turns with two crossing straight sections between them resembling a figure eight.

Next we added the sensors to the simulation. Two types of sensors were tested; infrared distance sensors and ambient light sensors. Both sensor types were tested in the two previous experiments and shown to work reliably. The sensor that yielded the best results in this experiment was the ambient light sensors. This is not due to the sensors them selves, but the experimental set up.

The distance sensors require walls to be built to be able to detect the edges of the road. This gives us a challenge when the robots drive outside of the road. We tried making the walls solid, but this resulted in the robots simply getting stuck on the walls and the algorithm could never really get the fitness gradient of the population that is required for evolution. The other thing we tried was to allow the robots to drive through the walls during simulation. The robots could still see the walls but the collision detection was turned off. The problem with this approach is that the distance sensors are unable to handle being inside a solid and the resulting output was the same as if the walls were hollow. The robots could not separate seeing the walls from the inside from seeing the from the outside and hence could not get out of the walls once they had stumbled in to them. This is bound to happen to even the best of the phenotypes in the first generations and the fact that the robots had to break the “rules” keeping them on the road in order to get back on the road when they inevitably wandered of haltered the progress of the evolution. The light sensors on the other hand works by returning the colour value of the overlay/background image at a given point relative to the robot. Having

no physical barrier to keep it on the road the network was able to teach it self to recover when accidentally driving out side of the road.

Two arrays of sensors positioned in two concentric half circles with 5 sensors in each array for a total of ten sensors was chosen as the inputs to the neural network. This configuration was inspired by Drchal et al. [2009]. The sensor configuration can be seen in figure 4.6

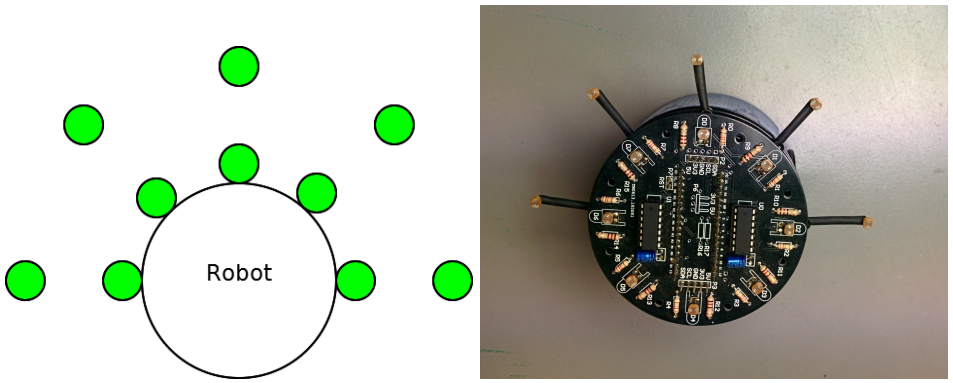


Figure 4.6: Left: Distribution of ambient light sensors around the front of the robot. Right: Sensor array implemented on the physical robot. (Robot seen from above. Front of robot towards the top of the image.)

The simulation of one generation consists of the following steps:

1. Place each robot in the population on a random section of road. Also turn the robot so it faces 45 degrees to the left or right of the direction of the road. This will reduce the effect of some robots being lucky with their placements on a straight piece of road.

2. Run the simulation giving each robot its own fitness using these rules:

   - For every new time step i, take the euclidean distance from the current position pos_i, to the position in time step $i-1$ and add it to the fitness of the robot. $fitness = fitness + distance(pos_i, pos_{i-1})$. This rule will encourage the robots to drive quickly. It will also punish any behaviour that causes the robot to slow down, like driving out side the road surface and crashing with other robots.
   - If the difference in speed between the left and right wheels of the robot is over 5, subtract 0.8 from the fitness in each time step. if$(Max(wheel_L, wheel_R) - Min(wheel_L, wheel_R) > 5) fitness = fitness - 0.8$. This rule stops the robot from exploiting the first rule by spinning quickly in a tight circle and getting a high fitness without really moving anywhere. It also encourages driving straight as much as possible.

3. if the robot travels outside of the road reduce the maximum speed of the robot by a factor of 6. This will indirectly punish the fitness of the robot by reducing the distance it is able to travel in one time step.

4. After a set number of time steps the simulation halts and the fitness of the individuals are stored in the phenotype. The thread running the GA then selects the set of parents based on their fitness and creates a new population using a combination of mutation and crossover to replace the old one.

5. The new phenotypes from the new population are inserted in to the existing robots, effectively replacing the old "brain" with the new. The fitness of the robot is reset to zero.

6. The simulation can now resume from step 1 with the new generation.

Several neural network topologies were tested, but we got the best results using a multi layer network with two hidden layers and full connection between layers. An illustration of the network topology can be seen in figure 4.7

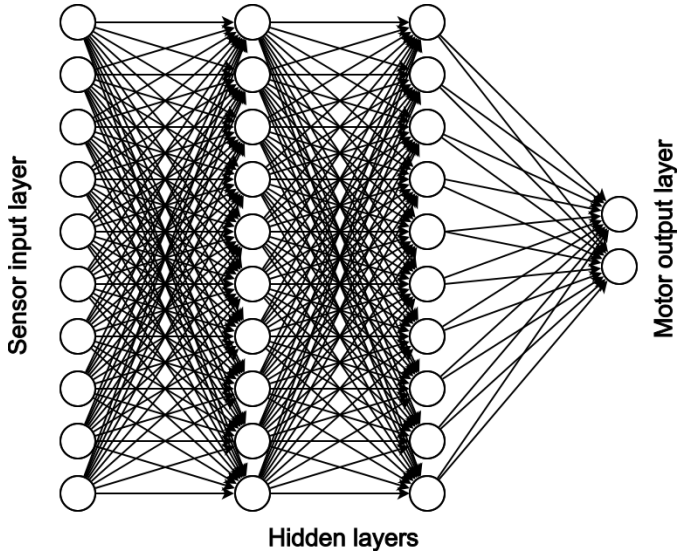


Figure 4.7: Full topology of the neural network network.

4.3.3 Results

The simulation is run until it is on the 1000th generation. A graph of a random simulation run can be seen in Figure 4.8. We can see from the graph that the population starts to stabilize around generation 450, however we also see that the standard deviation slowly decreases while the average stays high indicating that

the algorithm is still converging after a good solution is found. The GA parameter values used in this simulation run can be seen in Table 4.1.

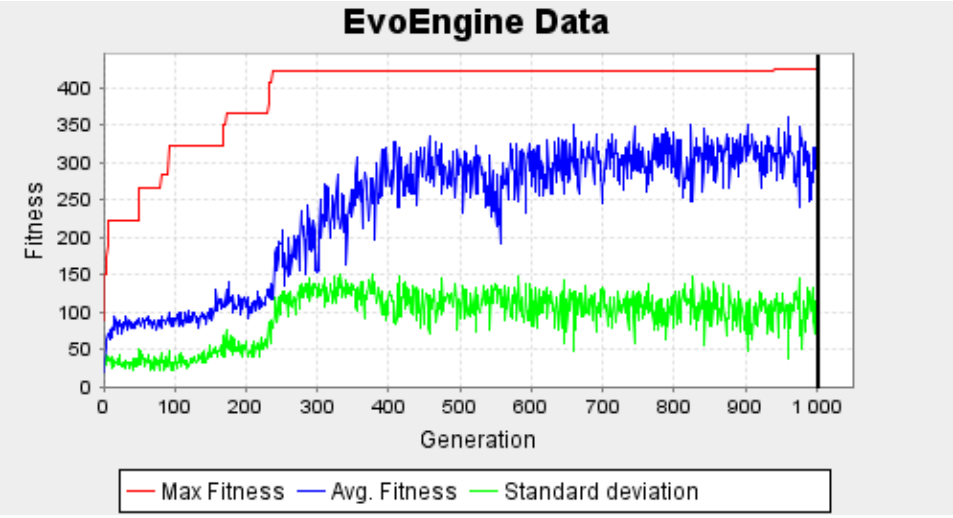


Figure 4.8: Graph showing a randomly selected simulation run. Top line is the highest fitness achieved. Middle line is the current average fitness. Lowest line is the standard deviation of the current population fitness distribution.

Mutation rate	0.6
Crossover rate	0.4
Population size	40
Selection mode	sigma scale

Table 4.1: Simulation parameter values

Several parameter values were tested until the GA could reliably find a stable solution within the allotted number of generation. Changes in these values affects the outcome of the algorithm in different distinct ways.

A high mutation rate in the population can ensure that we explore a larger part of the solution space and can make it easier to escape local maxima. The downside of having a high mutation rate is that we inhibit the ability of the population to hill climb to a good solution, effectively destroying a decent solution before it can become a great one. Figure 4.8 shows the standard deviation for the fitness over the population as the bottom graph. While this does not directly show the deviation in genome distribution we can still make some assumptions about the distributions based on this data. We can for instance observe form the graphs that after a partial solution is staring to emerge around generation 300 the average fitness of

the population climbs as the deviation between the individuals in the population slowly decreases.

It is worth noting that the reason that we had a low value for the standard deviation at the start of the experiment was not because the different genomes in the population were similar but rather because the different solutions were similarly bad since the genomes were initialized with random values. We observed as expected that with a high mutation rate the fitness distribution was generally high, and remained high as the population struggled to converge on the good solutions due to the large differences in the genome from one generation to the next. Conversely we observed that with a low mutation rate the standard deviation changed at a slow rate increasing the time need to find potential solutions often resulting in the population becoming stuck at sub par solutions lacking the momentum to escape local maxima.

The crossover rate can, in a similar way to the mutation rate, help the population escape local maxima by combining parts from two different solutions in to a new one. This requires that we have a distribution of different solution to select from. Having a high crossover rate increases the chances of a promising solution being suppressed in early generations by being paired with a worse solution. By having a low crossover rate we may have an increased risk that different solutions converge on a local maxima.

To ensure that one has a wide coverage of the solution space at the start of an experiment it is usually advantageous to have a large population size. In our experiment the population size is limited by the physical size of the arena. A small population size limited the solution space coverage at the beginning of the experiment resulting in poorer solutions and slower convergence. Having too many robots in the arena at the same time increased the probability of two robots colliding with each other. This is not a bad thing in it self, as we wanted to see if the robots could evolve a strategy to cope with such events, but with large populations the indirect penalty of colliding with other robots would drown out any progress a robot had made in navigating the road network, often resulting in solutions where the robots would rather get a lower score by driving in tight circles than risk getting stuck on another robot.

The selection mechanism is the method used to select which individuals to use when creating the next generation, and which to discard. Different mechanism have different effects on the way the algorithm converges and choosing the best mechanism is largely dependent on the way your fitness function is set up. Simply put, the methods differ in the way they penalize low fitness compared to high fitness. High penalty results in quick convergence but with a higher chance of missing good solutions or getting stuck on local maxima. Conversely a low penalty will have a slower convergence time. If the penalty is very low we run the risk of noise or random chance drowning out good solutions slowing down the search. We run a similar risk with a very high penalty where some bad solutions will get a high fitness by chance while some good solutions will be unlucky. The selection mechanism used in this experiment is Sigma scale selection. It works by assigning a probability of being selected for the next generation based on equation 4.1 where

P_i is the probability for selection, F_i is the individuals fitness and $\frac{F_p}{\mu}$ and σ is the average fitness and standard deviation over the population respectively.

Other selection mechanisms that were tested was Fitness proportionate selection where the probability for selection is directly proportionate to the fitness over the population average, tournament selection where groups of individuals are extracted and compared before selecting the best ones in each group and truncation selection where we simply select a fixed number of individuals with the highest fitness.

$$P_i = 1 + \frac{F_i - (\frac{F_p}{\mu})}{2 * \sigma} \tag{4.1}$$

When the simulator had produced a result we were happy with, we transferred the genome of the individual with the highest fitness in to an identical neural network running on the actual robot. To test the memory limitations of the ChIRP robot when using neural network controllers, we also created larger networks to see when the memory would run out. The largest network we tested had 49 neurons and 390 synapses. The network used in the experiment had 32 neurons with 220 synapses. It is probably possible to get considerably larger network running by disabling the serial communication library and improving the memory management. For other suggestions on how to get larger network to work, see the future work section in Chapter 5.

When we had transferred the controller in to the physical robot we created a new, smaller testing track for the robot to run on. This track was then projected on to the robot area from an over head projector. The same camera tracking software used in the second experiment was used to gather position data from the robot. A screenshot of the camera tracking software recording positional data is shown in Figure 4.9.

The recorded data was put back in to a program and overlayed on to the projected image. The recorded data can be seen in Figure 4.9. This shows the robots doing three laps around the course in two different directions. The darker coloured line is the tracking point. The lighter coloured line around it is the outline of the robot to show the actual size. As we can see from the data, the controller learns to drive on one side of the road to avoid collisions. The small intersection of the robots at the top left corner may be caused by the fact that the projected road has a slightly smaller diameter than the road use in the simulator. This occurs because of the parallax effect caused by the projector being placed at the edge of the robot area. We ran the algorithm with the robots able to pass trough each other to confirm that it was not simple convergence on a stable solution that resulted in this behaviour. When collisions were turned off this behaviour disappeared. Figure 4.10 shows the experiment being run.

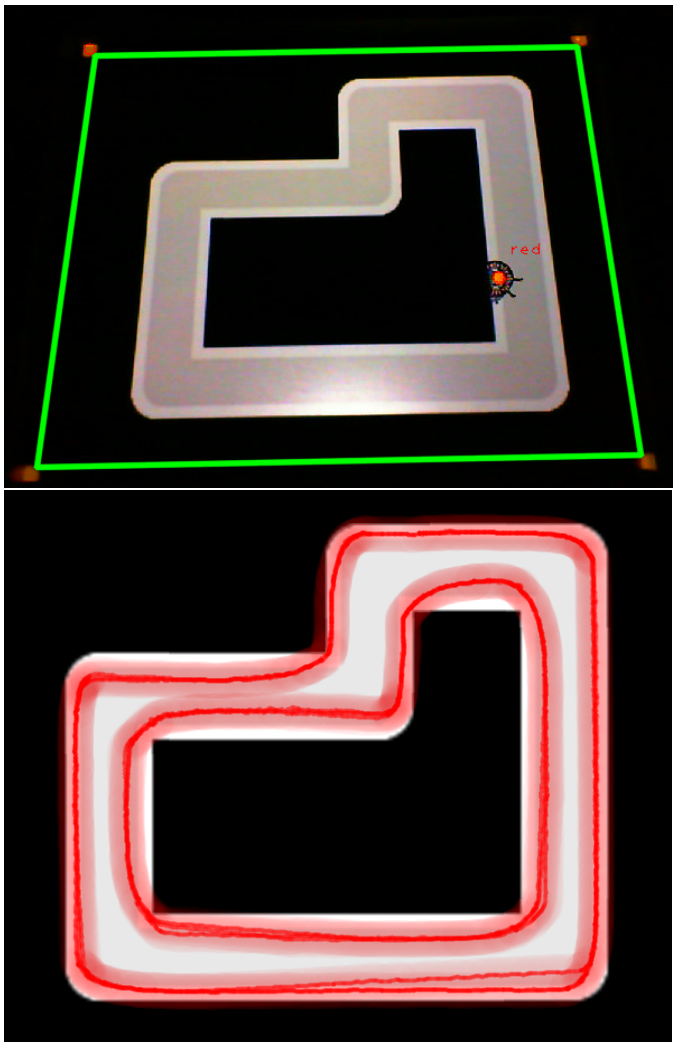


Figure 4.9: Top: Camera used to record the movements of the ChIRP robot. Bottom: Output from the recording overlayed on to the projected track.

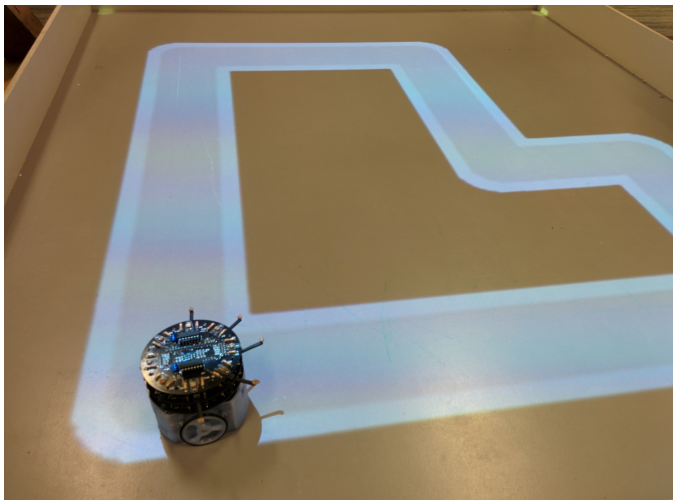


Figure 4.10: Experiment being run on the ChIRP robot.

Chapter 5

Evaluation and Conclusion

As we stated in the first chapter the goal of this thesis is to design and build a robot and then to evaluate the usefulness of the robot platform for use in AI research. While the experiments we have conducted would be possible to do on other robot platforms, we are not trying to make a comparison between the ChIRP robot and other platforms. However, our findings show that the ChIRP platform is a valuable research tool. We also hope that the approaches we have used can serve as inspiration for future robotic research.

The hardware and software we have tested and confirmed to be working include:

- *Infrared sensors* - Distance sensing and beacon tracking.
- *Bluetooth communication* - Two way communication between robot and both android tablet and computer.
- *Ambient light sensors* - Used to sense images projected from an over head mounted projector.
- *Camera tracking* - Tracking of robot positioning and rotation.
- *Simulator* - General purpose simulator created in Java.
- *GA and ANN libraries* - General purpose libraries for genetic algorithms and artificial neural networks.
- *General ANN implementation for ChIRP robot* - Artificial neural network controller for the ChIRP robot. (Supports an arbitrary number fully connected layers of arbitrary size.)

5.1 Evaluation

All of the experiments worked as expected with both the software and the hardware performing well during both the demonstrations and the lab experiment. The only challenge we had was keeping the robots running for several hours straight without

recharging. The upshot of having these long running demonstrations was that we inadvertently got to test the battery usage of the robots against the laboratory test conducted previously. The listed battery life for the ChIRP robots are 4 hours. This proved to be accurate with some time to spare in all of our experiments. We solved the problem by having a spare set of recharged robots on hand for when the demonstrations would last more than four hours. We could also have swapped batteries during the demonstration, but since we had enough spare robots it was simpler to just replace the ones that ran out of battery.

One other challenge we faced when using the robots was altering the code running on the separate sensor controller chips. Currently they have to be removed from the robot in order to be reprogrammed. We suggest two possible changes to alleviate this problem. The first is to extend the sensor libraries to include more functionality. This change will be the simplest to implement but the abilities of the sensor arrays will still be limited by features implemented. On the other hand, the current code is still far from using all of the available program memory on the ATtiny chips so many new features could be added with ease. The other solution is to add programming headers to the chips. This would however require a hardware change to the robots and new circuit boards would have to be made. One would still require an external programmer to program the chips, but the chips them selves would not have to be removed.

We suggest implementing both of these changes in to the next version of the robot software and hardware. A third option would be to run the sensor arrays using the main micro controller. The micro controller used in the ChIRP does not have enough analogue inputs however, so we would have to cut down on the number of sensors. We would also like to keep the available pins on the main controller free for the users of the robot to use for their own alterations. The final problem with this solution is that the sensor readings would have to be made on the main program thread and would therefore take up valuable computing time that is available to the user in the current layout. Hence we do not recommend this alteration.

Although the infrared sensors were tested thoroughly during the development of the robot, the real test of robustness and stability comes from actually working with the robots and performing experiments on them. In all of the experiments the infrared sensors worked flawlessly with high tolerances for ambient light both from natural and artificial lighting and with sensing distances of up to 15-20 centimetres with good resolution. Sensing of direct infrared light from a beacon worked over a considerate distance of about 40 centimetres or more.

Being able to control the robot remotely opens up many possibilities for larger and more complicated controllers that would otherwise not fit in to the memory of the robot CPU. The bluetooth communication worked without a problem with both low latency and good bandwidth. Being able to control the robot directly from a simulator can also help with cutting down development time.

Ambient light sensors are harder to use than infrared light sensors since they are sensitive to the visible light we use every day. This type of sensor can only be used if the lighting conditions are well known or under the control of the researcher.

In the human versus robot and the final validation experiments we were forced to remove some of the lights directly above the arena to make sure that the contrast between the light and dark areas of the projected image would be high enough. The light levels were not so low that the participants could not see what was going on. We see the combination of light sensors and an over head projector as a great way to simulate surfaces for the robot to sense as well as simulation of pheromones to name but two possible future uses.

The camera tracking system worked well for both reporting position data back to a simulator and for recording playback data. The trick to getting a stable tracking is to find a mate marker with an easily distinguishable colour. Once we had set the threshold values at the start of the experiment we did not have to readjust them again. This was true for both of the experiments where we used the system. We also set up two extra lights to ensure that the light coverage of the arena was even.

The simulator we developed is light weight and easy to interface with. With the possibility to add physical object using the physics engine we were able to make the robots behave closely to the way they behave in real life. When the controller was moved from the simulation over to the physical robot the behaviours observed in the simulator seemed to correspond very closely to the behaviours of the real robot. This observation was confirmed using the camera tracking system.

Having a general GA and ANN library was very helpful when designing the last validation experiment. The changing of parameters could be done by simply changing a value in the configuration of the library and running the experiment again. This ease of use is critical when you are testing out an algorithm to see what works and what does not.

When implementing the ANN controller on the ChIRP robot we tried to make it as general as we could while still keeping the memory usage as low as possible. Some trade off had to be made to the possible topologies that are supported by the controller. It only supports full connections between layers in the current configuration. However the number of layers and their size can be chosen freely as long as it does not exceed the memory restrictions on the micro controller. This will be discussed further in the future work section)

5.2 Contributions

I believe we have fulfilled our goals of designing and building an affordable modular robot and shown that the ChIRP robot is a viable cooperative AI research platform.

- **Research question 1 - Can we build a robot fulfilling all of the design requirements?** We have created a robot that is small in size, affordable and expandable with a long battery life. The software and hardware source files are available for anyone to use in their own project[1].

[1] ChIRP homepage: http://chirp.idi.ntnu.no

- **Research question 2 - Are the ChIRP robots sensors and actuators accurate and precise enough to be used in cooperative robot experiments?** The proof of concept experiments discussed in this thesis have tested the robot in far tougher environments that a robot usually faces in an lab. With the robots working well under hard to control lighting conditions, physical handling by the public and very long run times we would say that the robots have performed very well with none of the robots failing even once during any of the experiments. While the required accuracy and precision is dependent on the individual requirements of the experiments, our tests of the ChIRP design show them to be comparable to, or better than the other commercially available robot platforms we have tested.

- **Research question 3 - Is the microcontroller used in the robot powerful enough to be used in representative cooperative AI research?** This question is harder to give a definite answer to, since different experiments being run on the robot may have vastly differing requirements to the computational capabilities of the CPU. When we chose to use a Neural network as the controller in the validation experiment we did so because we wanted to test a controller that would require large amounts of memory. While we were able to exceed the limitations of the CPU in that particular use case, by increasing the network size, we suggest a method in the future work section of this chapter that could vastly increase the possible size of a neural network running on the platform. We may not be able to give a definite answer to whether or not the the microcontroller is powerful enough in all cases. We believe however, that we have shown that it is indeed powerful enough to be used in many, if not most cases. As a final note on this answer we want to mention that the expandable nature of the ChIRP platform may actually allow for an unrestricted amount of processing power if the main microcontroller is used as a secondary processor. This can either be accomplished by adding an additional, more powerful controller as a hardware extension or by using the bluetooth extension to relay commands from an external computer as we demonstrated in the platooning experiment.

5.3 Future Work

We have shown that the ChIRP robot platform can be a valuable tool for cooperative robotic research, and that the platform can help future researchers to create novel experiments in robotic AI in the future. We look forward to seeing future research projects take inspiration from some the techniques we have demonstrated to do further novel experiments.

Through out the experiments we have collected some suggestions for improvements that could be made to the the systems that we have demonstrated.

The first improvement would be to add some additional features to better control the sensor array without having to reprogram the controllers. We would also suggest adding programming headers to the controllers to make them easier to

program.

We suggest upgrading the bluetooth modules to v4.0 for lower power consumption and additional features. When the extension was created, there was few bluetooth v4.0 modules available for a good price. The price and availability of these modules have improved greatly since then and the alteration is now trivial to make.

We would suggest adding a colour sensor to the ambient light sensor array. This would make the projector use case even more versatile than it already is.

By redesigning the robot electronics to use surface mounted component we could gain several advantages. The external boards such as the power circuitry, arduino micro and the motor controller could be placed on a single board. This would make the robot simpler and cheaper to manufacture as the assembly could be performed by a pick and place machine. This alteration is being done but is not completed at the time of writing this thesis.

Many changes could be made to improve the usability of the simulator. We suggest adding a proper graphical user interface for controls instead of the keyboard interface it uses now. A new layer of interfaces should also be added to improve usability.

The GA and ANN libraries should be extended to include a wider range of properties. There are a wide range of available options available now, but having more would make the libraries even more useful.

The final suggestion is for the ANN controller implementation on the ChIRP robot. We tested the current code to see how large a network we could fit in to the memory of the micro controller. While this network is large enough for many applications, we suspect that a much larger network could fit in to memory if we used the PROGMEM feature of the AVR micro controllers[2]. This feature in short allows a program to store and use read only variables in program memory. The code used in the validation experiment only use about half of the program memory (not optimized and could probably be using far less) leaving a very large amount of memory to be used to store the network weights. We hope this possibility will be explored in the future as it has the potential to extend the supported network size greatly.

[2]Application note from ATMEL: `http://www.atmel.com/Images/doc8453.pdf` (section 3.5)

Bibliography

Adams, D. (1979). *The Hitchhiker's Guide to the Galaxy*. Pan Books.

Aristotle (350BC). *Physicae Auscultationes*.

Berg, J. and Karud, C. H. (2011). Swarm intelligence in bio-inspired robotics. Master's thesis, Norwegian university of science and technology.

Bergenheim, C., Shladover, S., and Coelingh, E. (2012). Overview of platooning systems. In *Proceedings of the 19th ITS World Congress*. Chalmers.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2.

Darwin, C. (1859). *On the origins of species by means of natural selection*. John Murray, London.

Drchal, J., Koutník, J., and Snorek, M. (2009). Hyperneat controlled robots learn how to drive on roads in simulated environment. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1087–1092. IEEE.

European Commission (2015). `http://ec.europa.eu/research/researchersnight/index_en.htm`. (Accessed on: 2015-04-24).

Garnier, S., Gautrais, J., and Theraulaz, G. (2007). The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.

Marapane, S., Holder, M., and Trivedi, M. (1994). Coordinating motion of cooperative mobile robots through visual observation. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 1994. Humans, Information and Technology., 1994*, volume 3, pages 2260–2265. IEEE.

Maslow, A. H. (1943). A theory of human motivation. *Psychological Review*, 50(4).

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

McLurkin, J., Lynch, A. J., Rixner, S., Barr, T. W., Chou, A., Foster, K., and Bilstein, S. (2013). A low-cost multi-robot system for research, teaching, and outreach. In *Distributed Autonomous Robotic Systems*, pages 597–609. Springer.

Minsky, M. and Papert, S. (1969). Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, 19:88.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT press.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco.

Norwegian road administration (2015). `http://www.vegvesen.no/Fag/Fokusomrader/Forskning+og+utvikling/Teknologidagene`. (Accessed on: 2015-04-24).

Rojas, R. (1996). *Neural Networks, A Systematic Introduction*. Springer-Verlag.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5.

Skjetne, C., Haddow, P., Rye, A., Schei, H., and Montanier, J.-M. (2014). The chirp robot: A versatile swarm robot platform. In Kim, J.-H., Matson, E. T. ., Myung, H., Xu, P., and Karray, F., editors, *Robot Intelligence Technology and Applications 2*, volume 274 of *Advances in Intelligent Systems and Computing*, pages 71–82. Springer International Publishing.

Stølsvik, J. T. and Utgaard, N. (2014). Investigating the effect of a robotic presence compared to a virtual robot in teaching angles and turn measurements to children. Master's thesis, Norwegian university of science and technology.

Zakir, U., Leonce, A., and Edirisinghe, E. (2010). Road sign segmentation based on colour spaces: A comparative study. In *Proceedings of the 11th IASTED International Conference - Computer Graphics and Imaging (CGIM 2010)*. ACTA Press.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In Şahin, E. and Spears, W., editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer Berlin Heidelberg.

Appendices

5.4 Appendix A - Existing robot platforms

Table 5.1 shows an overview of other available robot platforms and various technical data associated with the platforms.

	E-puck	**Khepera III**	**Jasmine**	**Kilobot**	**S-Bot**
Size	d:70mm h:50mm	d:130mm h:70mm	30mm^3	d:33mm h:34mm	d:116mm h:100mm
Locomotion	wheeled, 13cm/s	wheeled, 50cm/s	wheeled	vibration, 1cm/s	wheeled
processor	DsPIC 30F6014A @ 40MHz	DsPIC 30F5011 @ 60MHz	ATmega88 @ 20MHz	ATmega 328 @ 8MHz	XScale @ 400MHz
Battery	LiION, 3.6V 1.4Ah	LiPo, 1.35Ah	2x LiPo caps, 0.25Ah	LiION, 3.4V 0.16Ah	LiION, 10Wh
Battery Life	1-2 hours, full speed	8 hours, with no MC	1-2 hours	3-24 hours	1-2 hours
Sensors, actuators	camera, light, IR distance, micro-phone, accelerom-eter	IR distance, light, ground distance, ultrasonic sensors, WiFi	IR distance, bearing, colour light	distance light	gripper arm, IR, light, force, torque, accelerom-eter, tempera-ture, humidity, speaker, micro-phone
Price	1300 USD	3500-4000 USD	130 USD	15 USD	unknown

Table 5.1: Overview of other available robot platforms

5.5 Appendix B - Assembly instructions for the ChIRP robot

This appendix includes full assembly instructions of the ChIRP robot. In addition to being of use when building the robot, it also gives a detailed description of how the robot is designed and constructed.

Chirp v.1

Assembly instructions

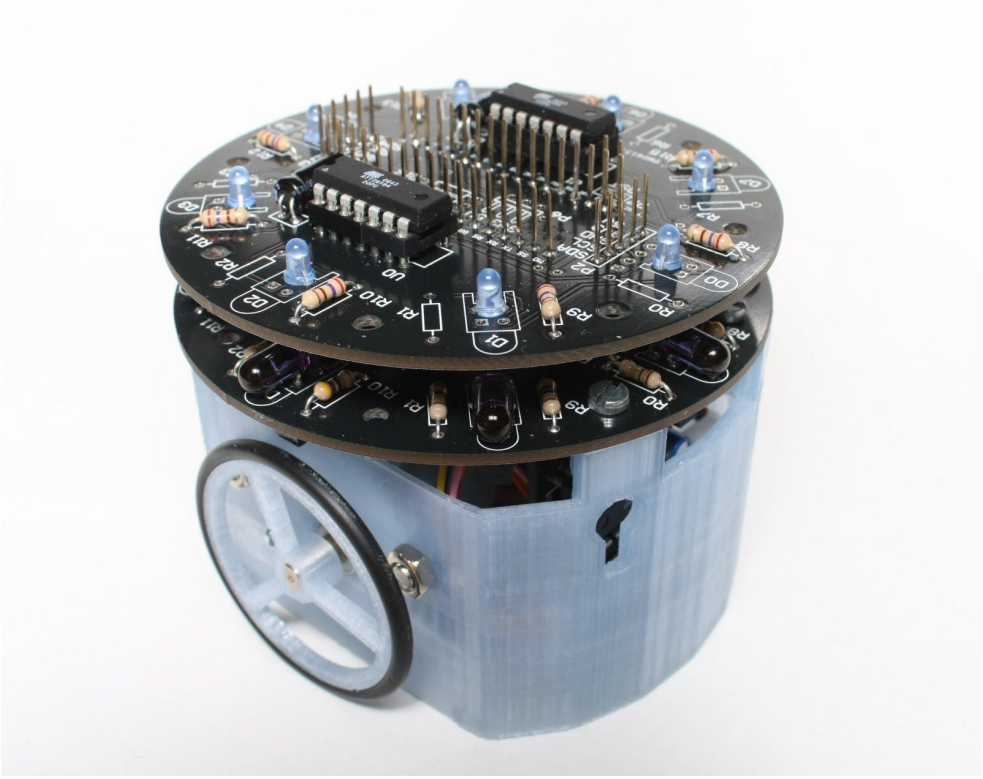


Table of contents:

Tools

The following tools are required for assembly:

- Small flat head screwdriver
- Wire snips
- Needle nosed pliers
- Soldering iron
- Solder
- Thin stranded wire

Parts list

Part	Amount	Placement
47 ohm Resistors	8x	R8,R9,R10,R11,R12,R13,R14,R15
100k ohm Resistors	8x	R0,R1,R2,R3,R4,R5,R6,R7
IR LED TSAL 6100	8x	D0,D1,D2,D3,D4,D5,D6,D7
Photodiode BPW 41 N	8x	S0,S1,S2,S3,S4,S5,S6,S7
ATtiny 84 program: Sensor front	1x	U0
ATtiny 84 program: Sensor back	1x	U1
ATtiny 84 program: Motor driver	1x	motorboard: U0
ULN2803 darlington driver IC	1x	motorboard: D1
14 pin DIP IC sockets	3x	U0, U1 and motorboard: U0
18 pin DIP IC sockets	1x	motorboard: D1
Capasitor 33uF	3x	C0, C1 and motorboard: C0
Capasitor 1000uF	1x	motorboard: C1
17 pin stacking header	2x	P0, P1
5 pin male header	5x	P2, P3 and motorboard: P0, P1, P2
2 pin male header	2x	P5, P7
5 pin female header	1x	P4
2 pin female header	1x	Used to connect the power cell to the sensor board
Sensor board PCB	1x	
Motor driver board PCB	1x	
Arduino micro	1x	Into P0 and P1 (usb towards S4)
28BYJ-48 Stepper motors	2x	
LP-805060 2700mAh 3.7v LiPo battery	1x	
SLS12104 slide switch spdt	1x	

M4 screw 5mm	4x	Motor mounts
M3 screw 10mm	4x	Sensor board
M3 screw 5mm	2x	Power Cell
M2 screw 5mm	2x	Power switch
M4 nut	4x	Motor mounts
O-ring 34mm	2x	Around wheels
Robot plastic chassis	1x	
Robot wheels	2x	
PRT-11231 Power Cell - LiPo Charger/Booster	1x	

Assembly instructions

We will start the assembly with the soldering of the Sensor board and the motor board PCBs. The assembly should be performed on a working station with a fume extractor.
To protect the eyes from injury during soldering and trimming of components leads, protective glasses should be worn at all times.

Sensor board PCB assembly

Tools required

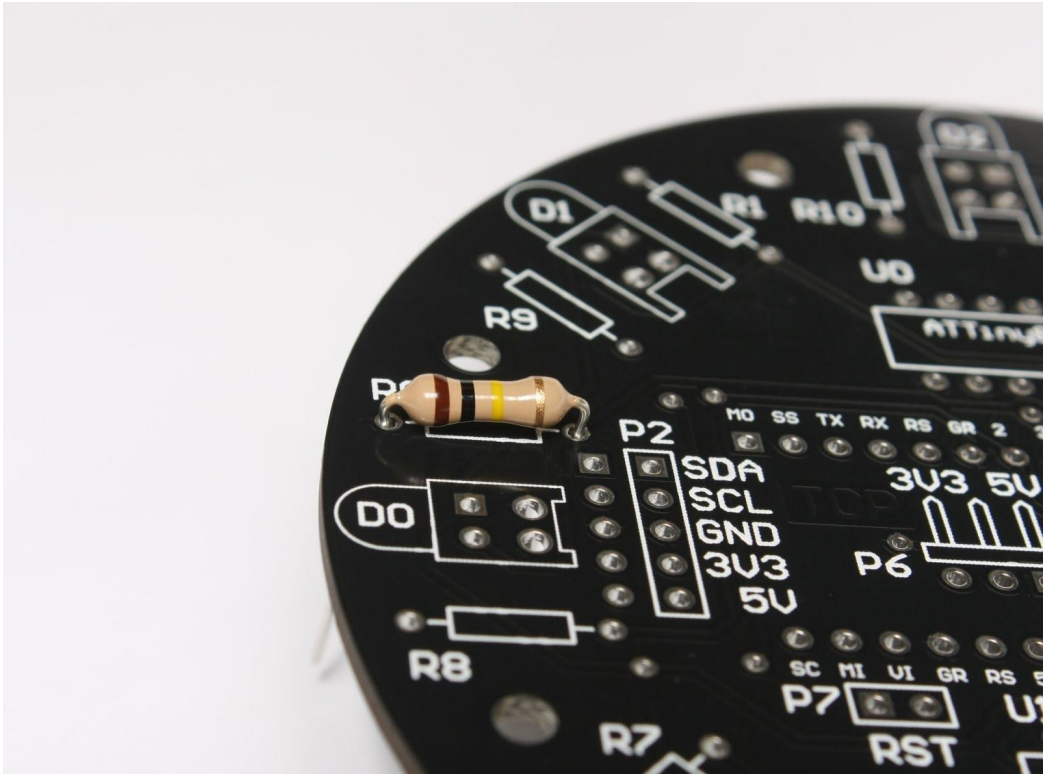
- Wire snips
- Needle nosed pliers
- Soldering iron
- Solder

Assembly

1. Resistors

The resistors are the easiest components to solder since you don't have to care about the polarity of the components; they can be soldered in either way.

Start by soldering the 100k ohm resistors at R0,R1,R2,R3,R4,R5,R6,R7. 100k ohm resistors are color coded with brown, black and yellow. Bend the leads and put them through the holes at the position marked R0. The component should be on the side of the PCB that has the little drawing of a resistor on it. This side will be referred to as the top side of the PCB.

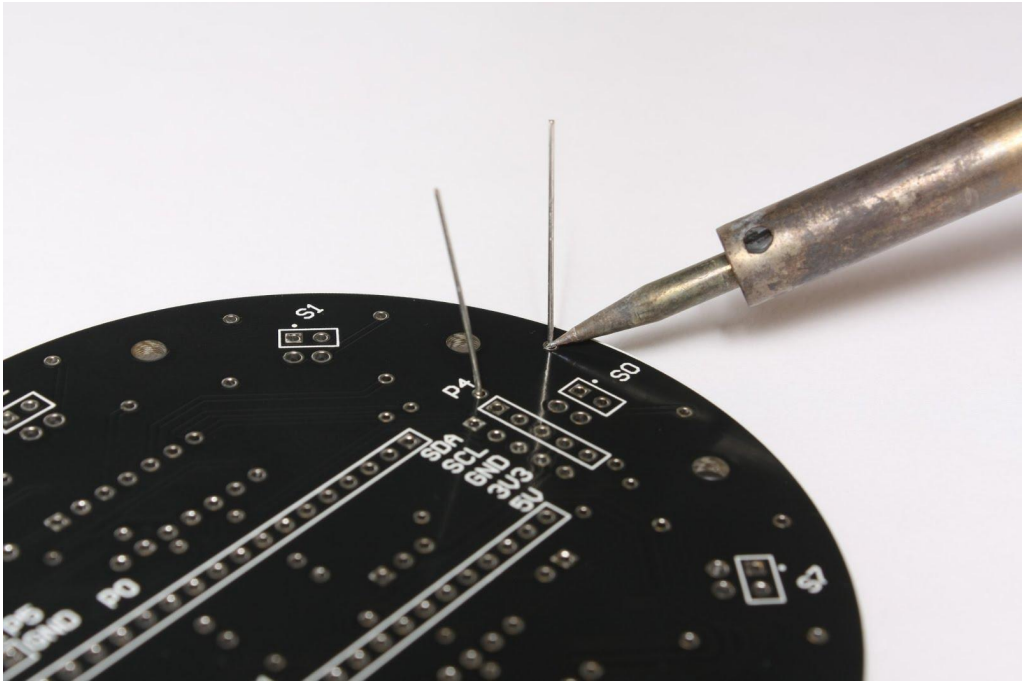


IMG1 100k ohm resistor placed at R0.

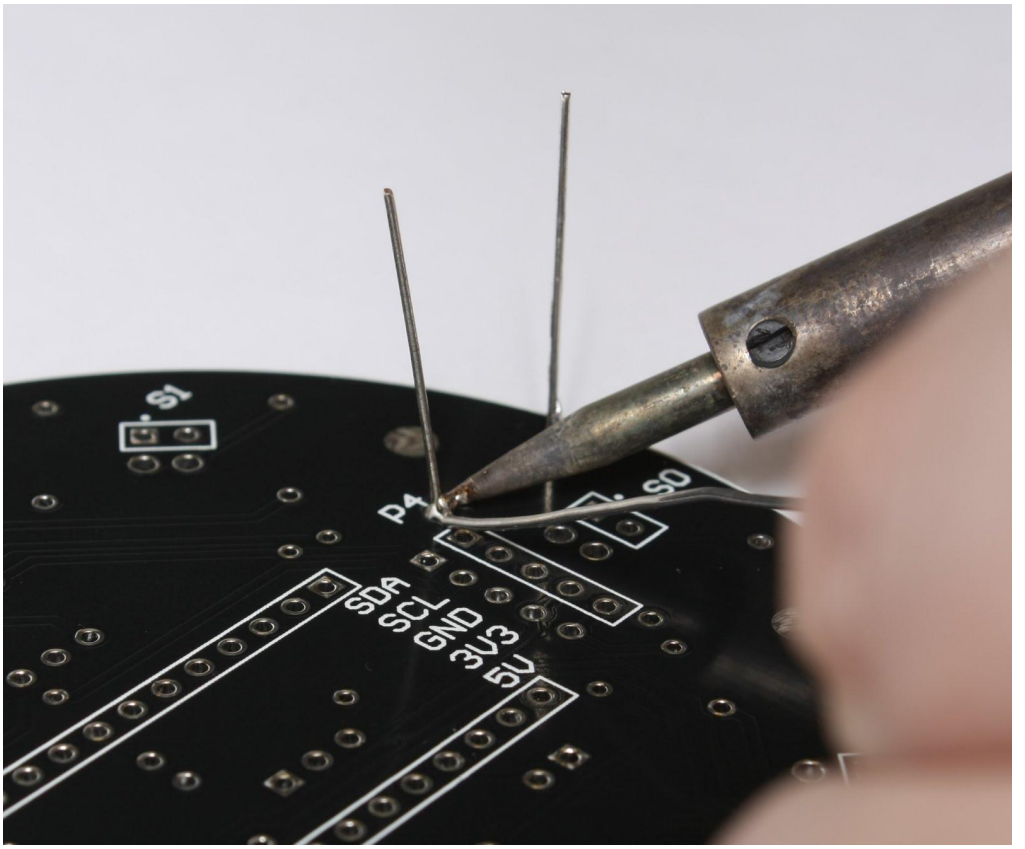
If you bend the leads out from each other on the other side of the pcb, they will hold the component in place.

Place your soldering iron at the base of the component leads on the bottom of the pcb so that it heats up both the silver ring on the pcb, and the component lead. The little silver ring around the hole is called a solder pad. After 1-2 seconds they should be hot enough to melt solder. Take a piece of solder and touch the solder pad on the reverse side of the lead from where the soldering iron is. The solder should melt instantly and form a little “volcano” engulfing the pad and going up the lead. If the solder is not forming nicely around the lead you may have to heat the lead and the solder pad more.

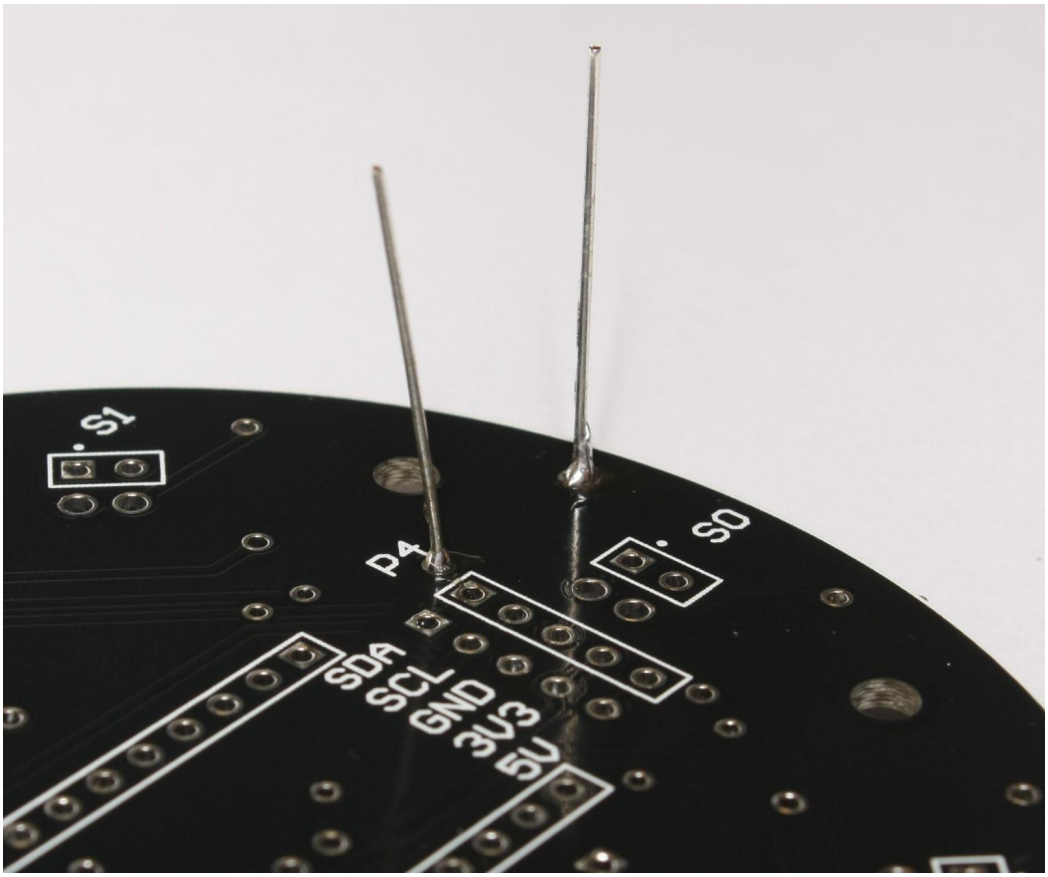
If you put too much solder on the pad, you can use a solder wick to remove some. to do this, reheat the solder with the soldering iron, touch the solder with the solder wick and remove the wick before removing the iron. The wick should have sucked up some of the solder.



IMG2 100k ohm resistor placed at R0, Other side of the PCB. Soldering iron placed at the base of the lead, contacting the solder pad on the pcb.

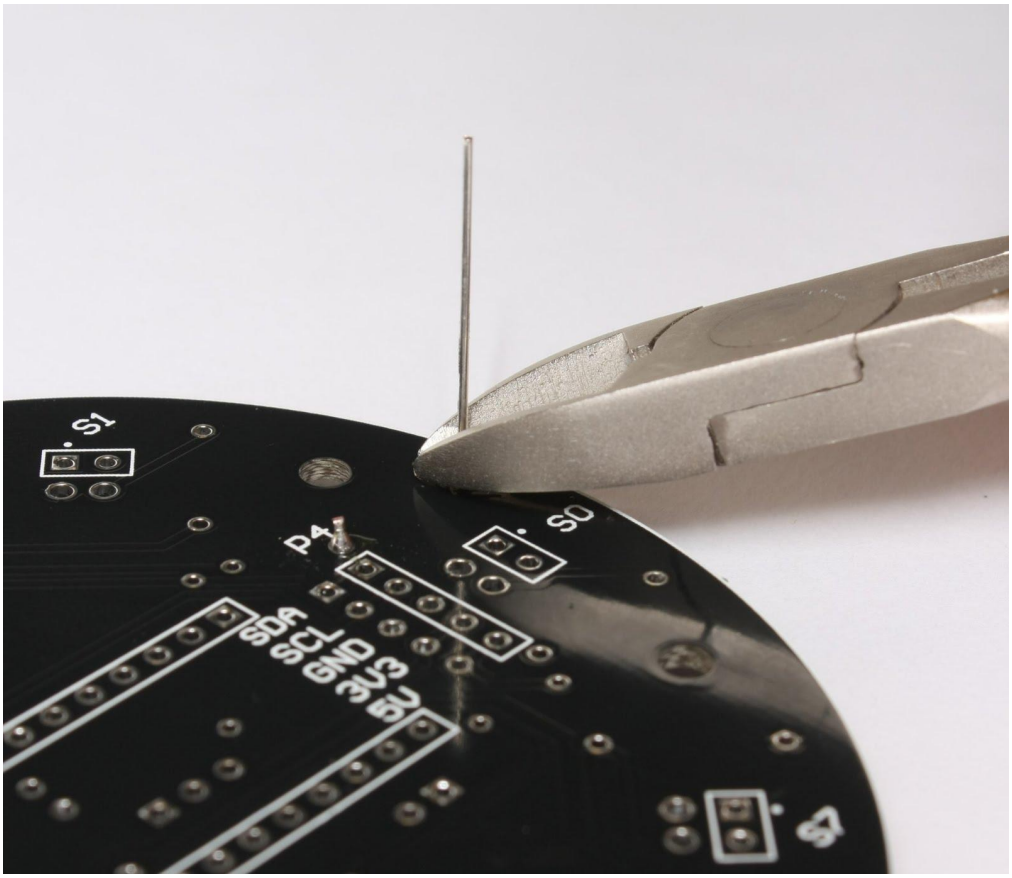


IMG3 The solder is starting to form a little volcano shape around the pad and lead.



IMG4 finished soldering the component.

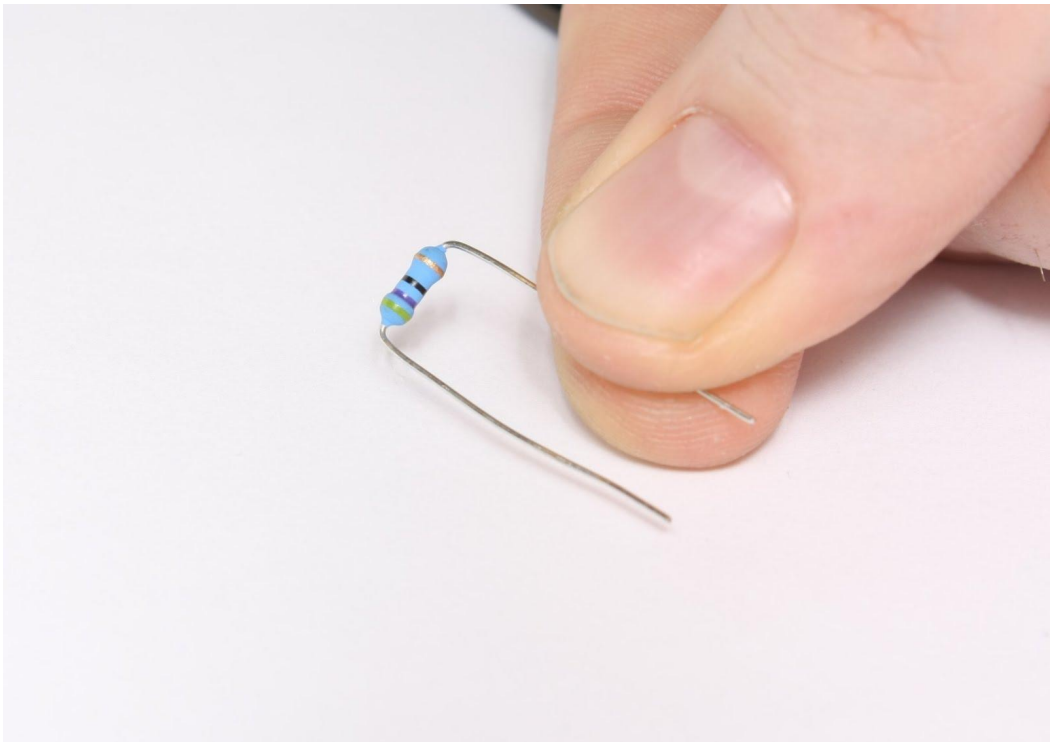
Once you have finished soldering the component you can use the wire snips to trim of the leads. Trim the lead as short as possible without cutting into the solder.



IMG5 trimming the leads

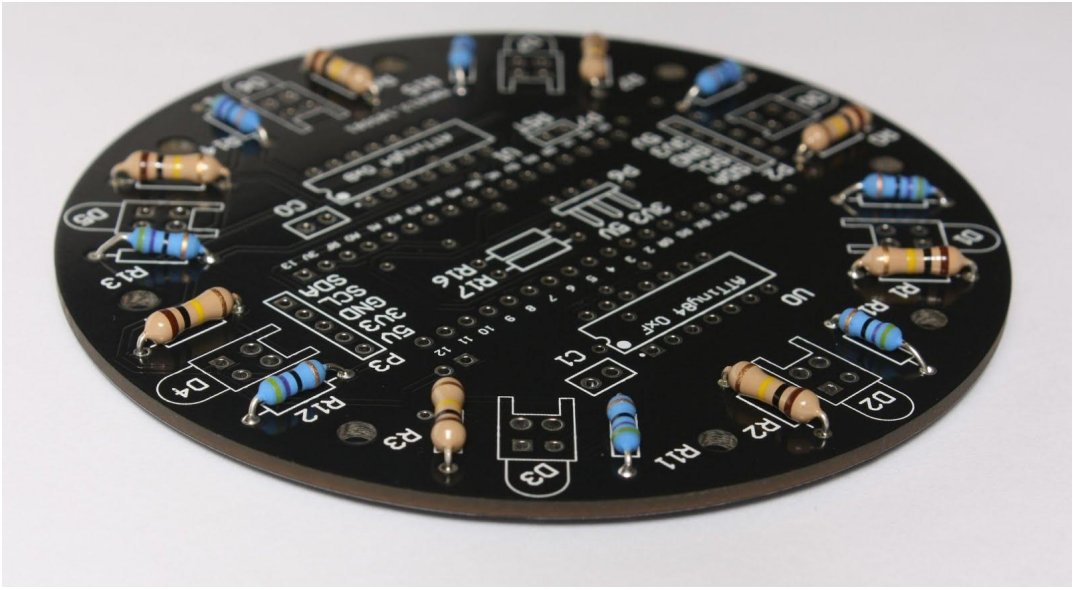
To save time you may wish to place several components first, and then solder all of them in one go. Please make sure you feel comfortable with all the steps before doing this.

After all of the 100k resistors have been soldered in place, it is time to place the 47 ohm resistors. Solder in the 47 ohm resistors at R8,R9,R10,R11,R12,R13,R14,R15. 47 ohm resistors are color coded with yellow, violet and black.



IMG6 47 ohm resistors

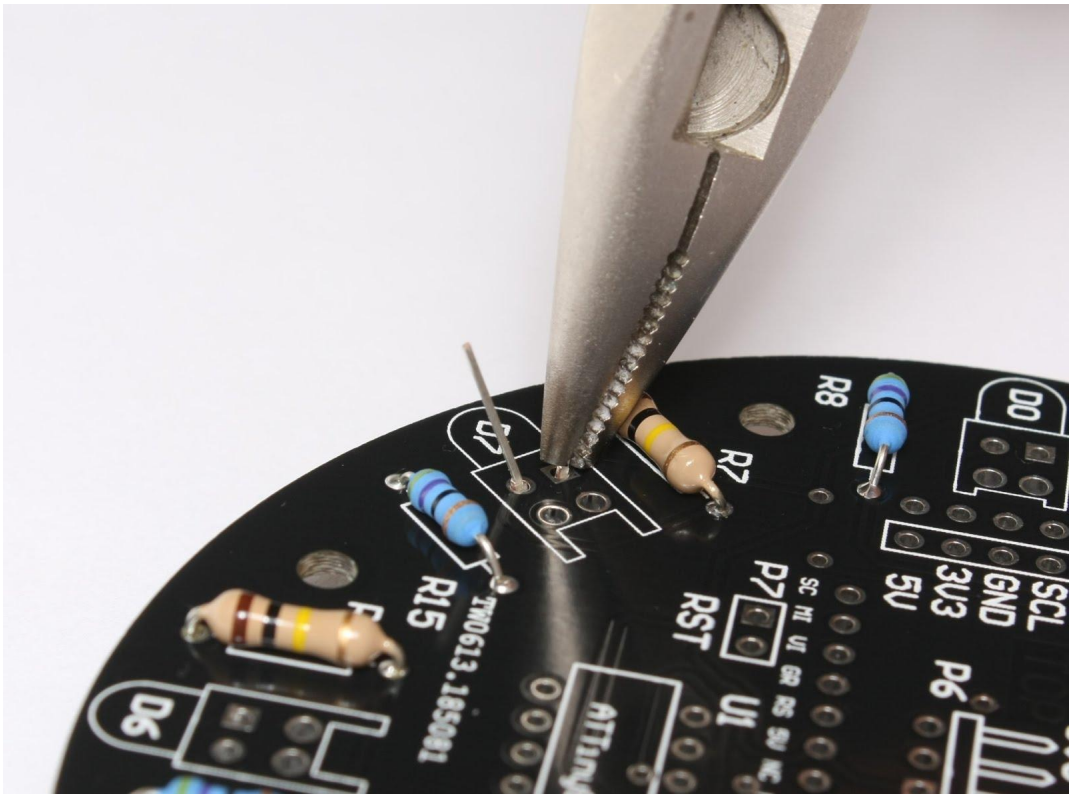
By now you should hopefully feel more comfortable with soldering, so we will skip the detailed instructions on that for this step. If you need to refresh the steps, they are luckily exactly the same for these components as they were for the 100k ohm resistors in the previous step so you can quickly flip back to look at them.



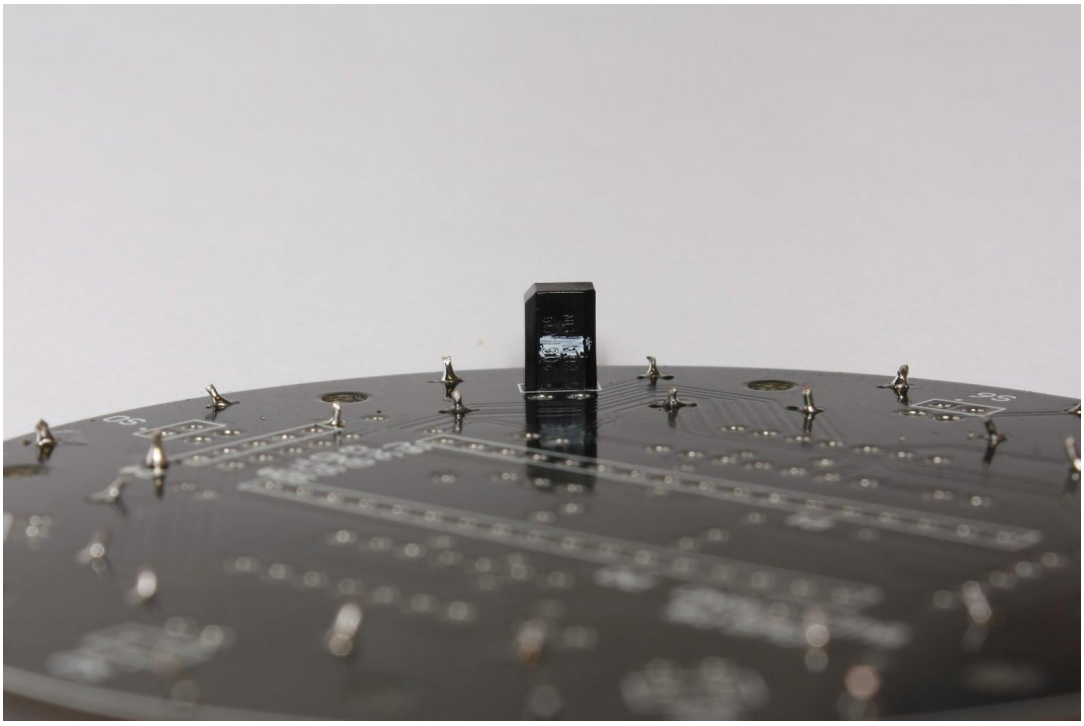
IMG7 After all of the 47 ohm resistors are solder in place the pcb should look like this.

2. Infrared receivers

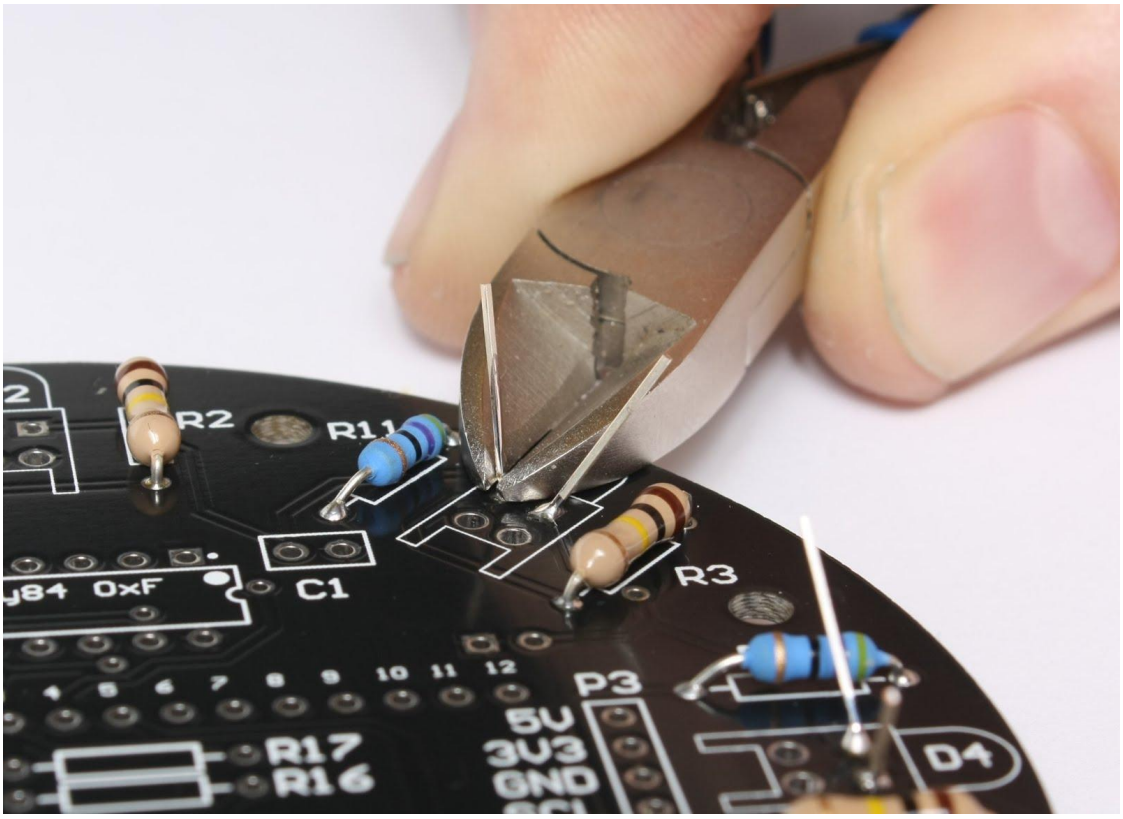
Now we move on to another type of component; the infrared receivers. The ir receivers look like small black boxes with two leads coming out of one of the short ends. The component also has a white mark on one of the larger sides. This mark indicates the back of the receiver. Unlike the resistors, the orientation of this component is very important. The side with the white mark on it should point to the center of the pcb. The receiver has to be mounted on the bottom of the pcb so the leads stick out on top. The receivers should be placed in the slots marked S0,S1,S2,S3,S4,S4,S5,S6,S7. The receiver should be soldered in place as close to the pcb as possible, so using a pair of pliers, bend the leads from each other as close to the pcb as possible.



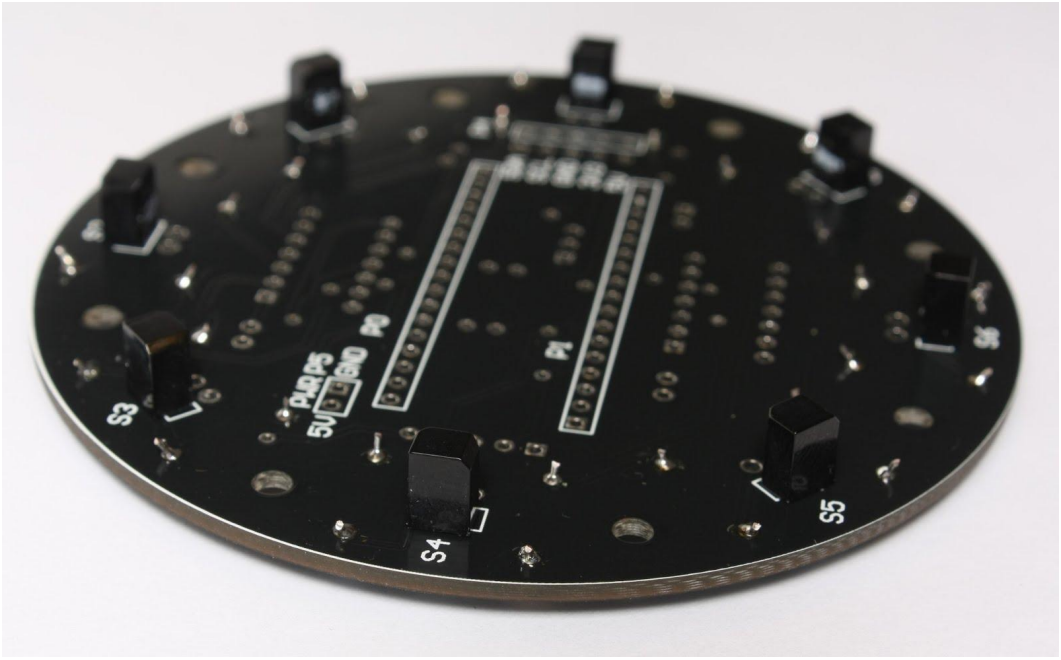
IMG8 bending the leads holds the component close to the pcb.



IMG9 the bottom of the receiver is held flush with the pcb with no room to wiggle.



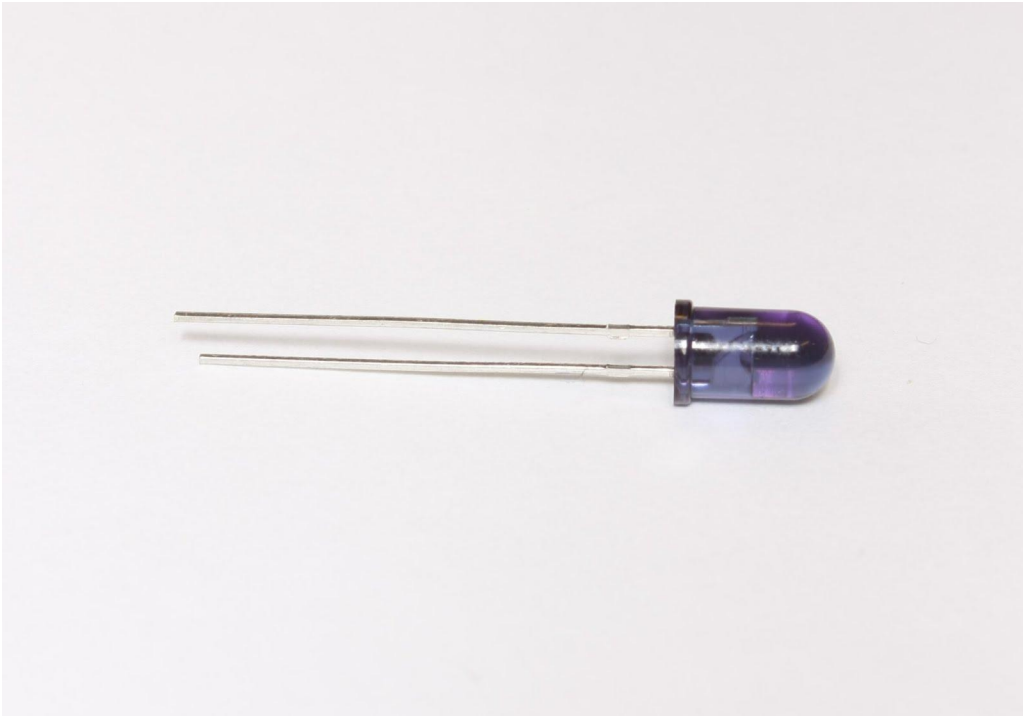
IMG10 trimming these leads as short as possible is very important as the infrared emitters will be placed on top of these.



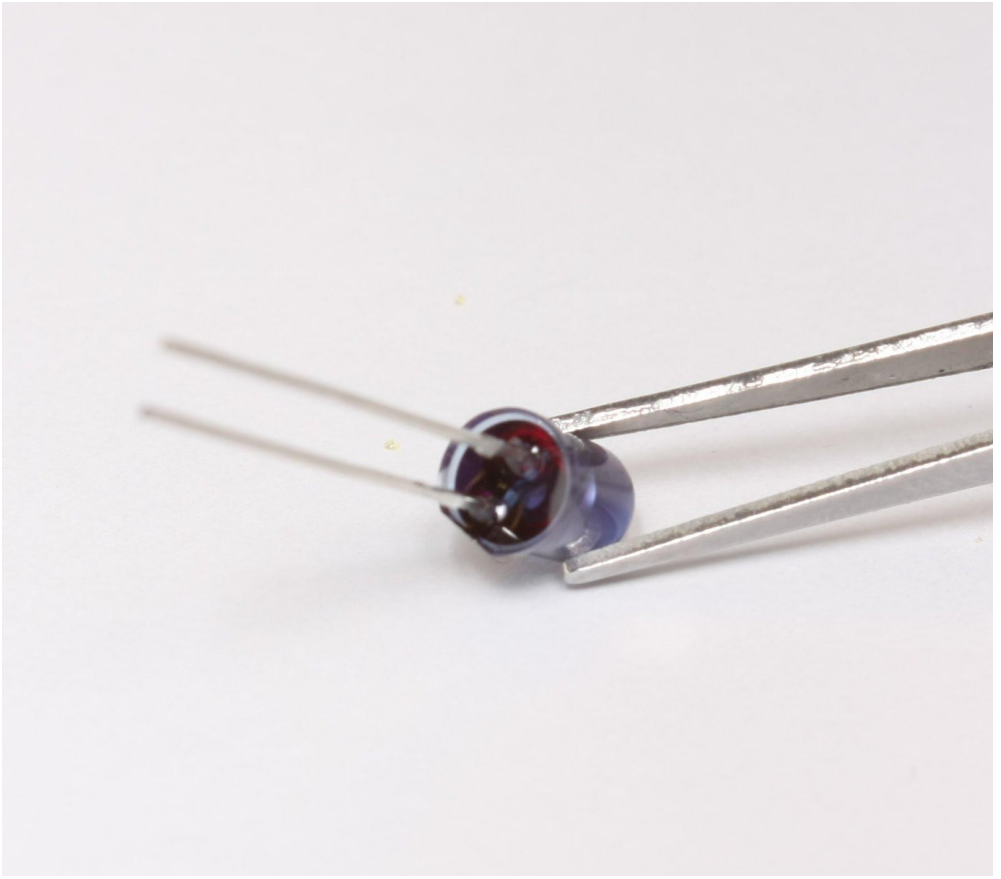
IMG11 the bottom of the pcb should look like this when all of the receivers are soldered in place.

3. Infrared Emitters

As with the infrared receivers, the orientations of the infrared emitters are very important. If you look at the picture (IMG12) you will see that one leg of the component is longer than the other. The long leg is the positive leg, and the short leg is the negative. The negative side of the “bulb” part of the emitter also has a flat part on its base (see IMG13).

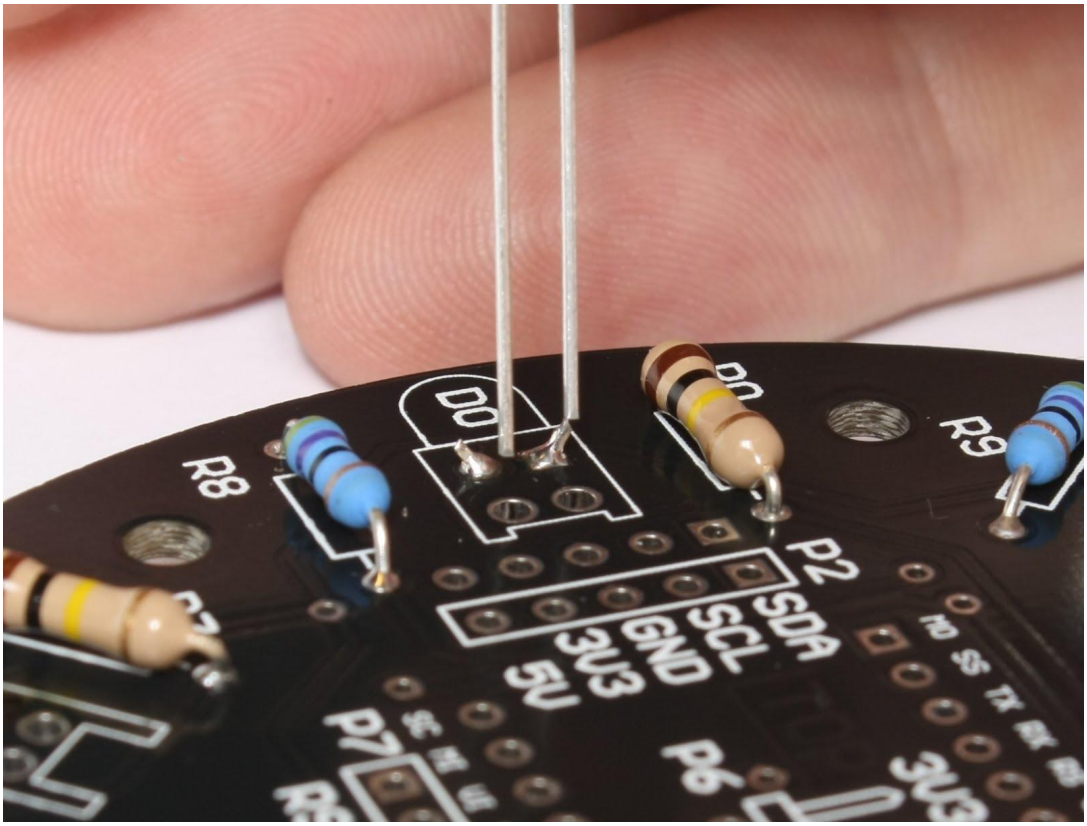


IMG12 Infrared Emitter LED the long leg is positive, and the short leg is negative.



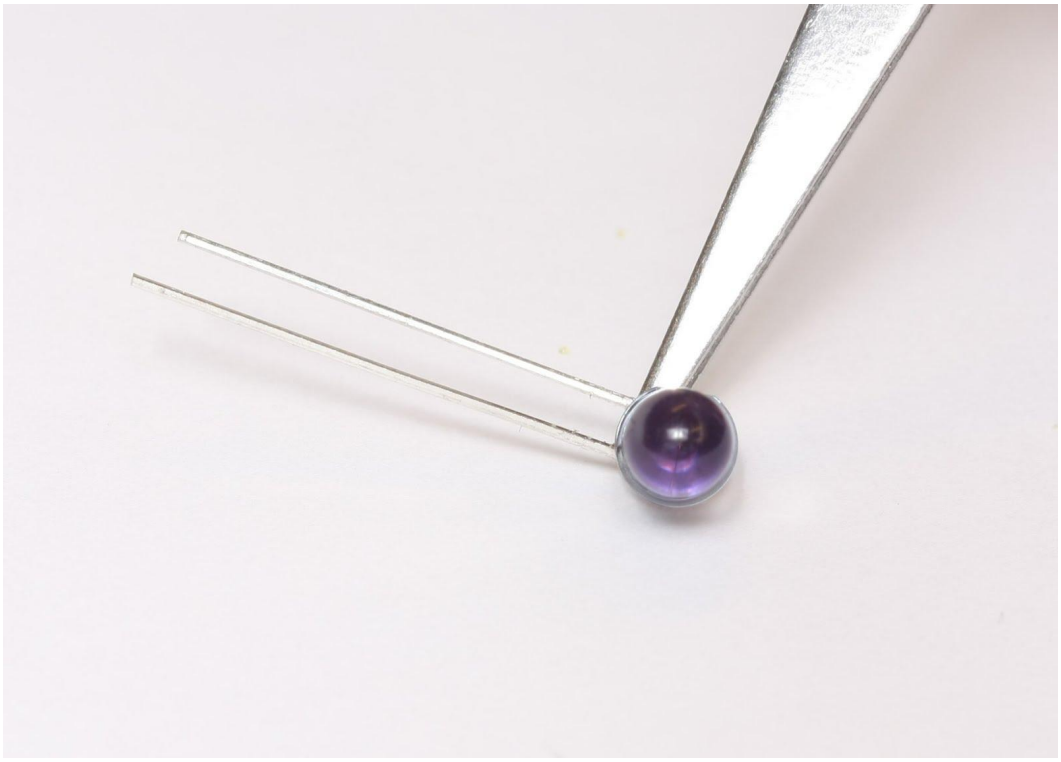
IMG13 Infrared Emitter LED the flat part of the bulb indicates the negative lead side.

The ir emitters should be placed in the marked D0,D1,D2,D3,D4,D5,D6,D7. with the positive long leg down the left hole (when looking from the center of the board and out see picture IMG14)

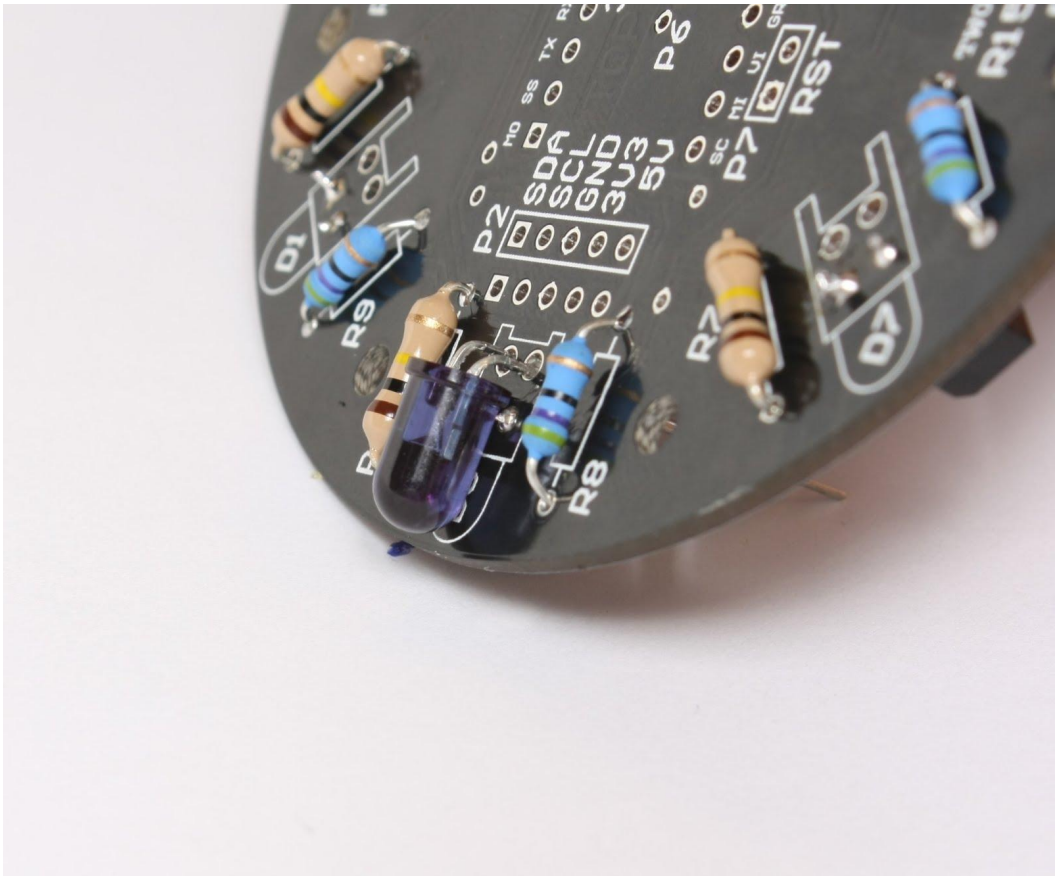


IMG14 positive long leg down the left hole

But before we place the emitters, we have to bend them. The infrared beam should radiate outward from the robot, so we have to make a 90° bend in the leads. Since the components have to have to be placed correctly on the pcb, we have to bend the leads the correct way. Grab the leads with a pair of pliers about 2-3 mm from the bulb. now make a 90° bend making sure that the positive long lead is to the left when the leads are pointing up, and you are looking straight into the bulb (see IMG15). The best way to make sure you are bending the legs the correct way is to lay the pcb down on a table, take the emitter with the pliers, align the correct leg with the correct hole on the pcb and bend so that the emitter is pointing out from the center of the pcb.

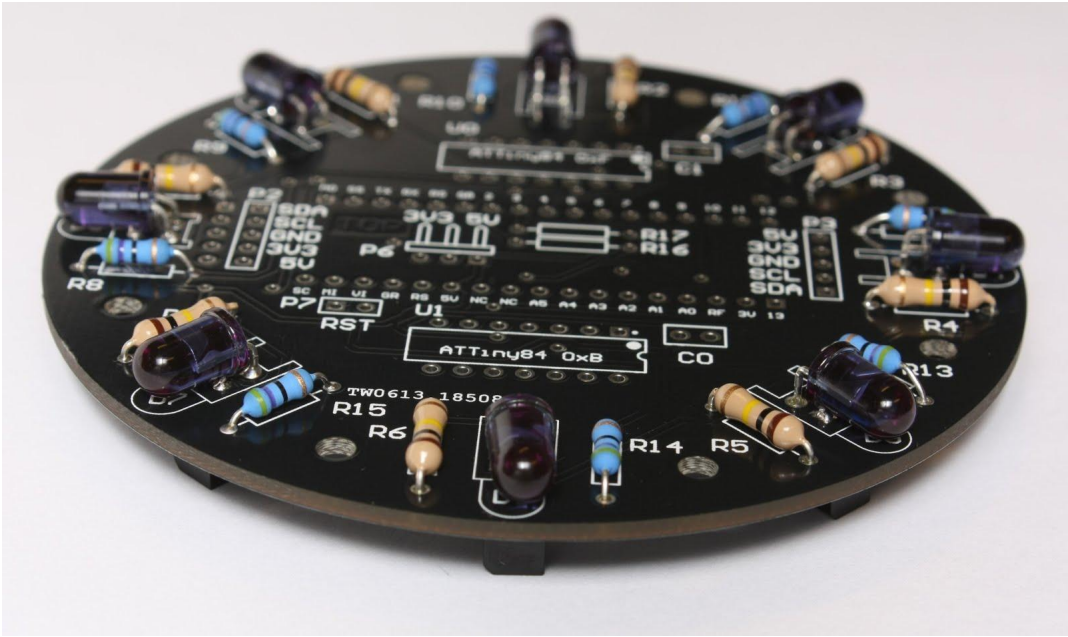


IMG15 ir emitter with the correct 90° bend



IMG16 ir emitter correctly placed.

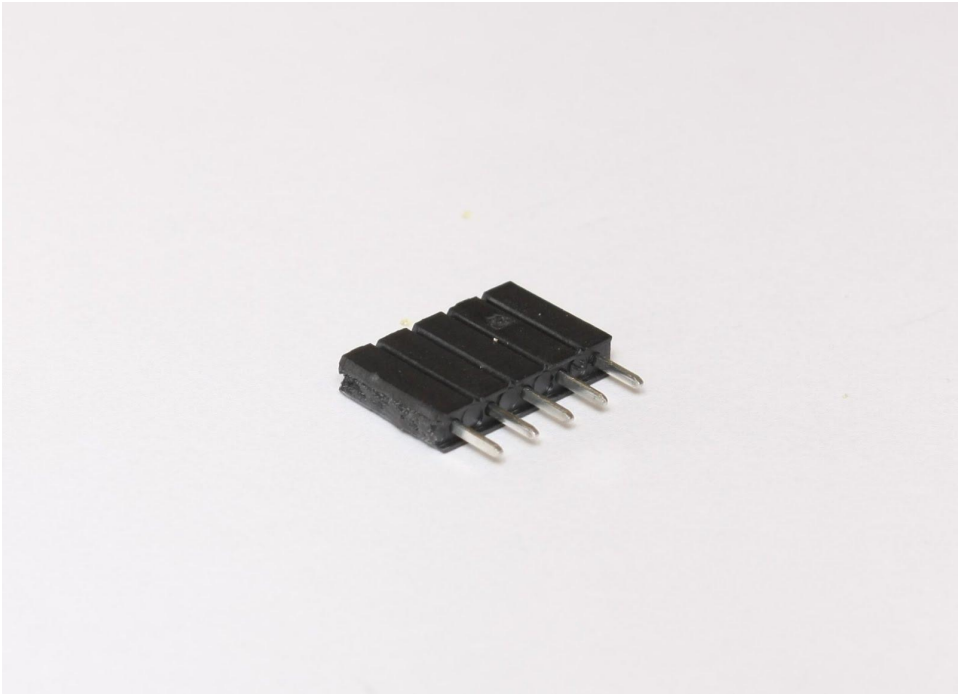
Once you have placed the ir emitter correctly and made sure that the correct lead is in the correct hole, spread the leads on the bottom side of the pcb and solder in place.



IMG17 when all of the emitters are soldered in place the board should look like this.

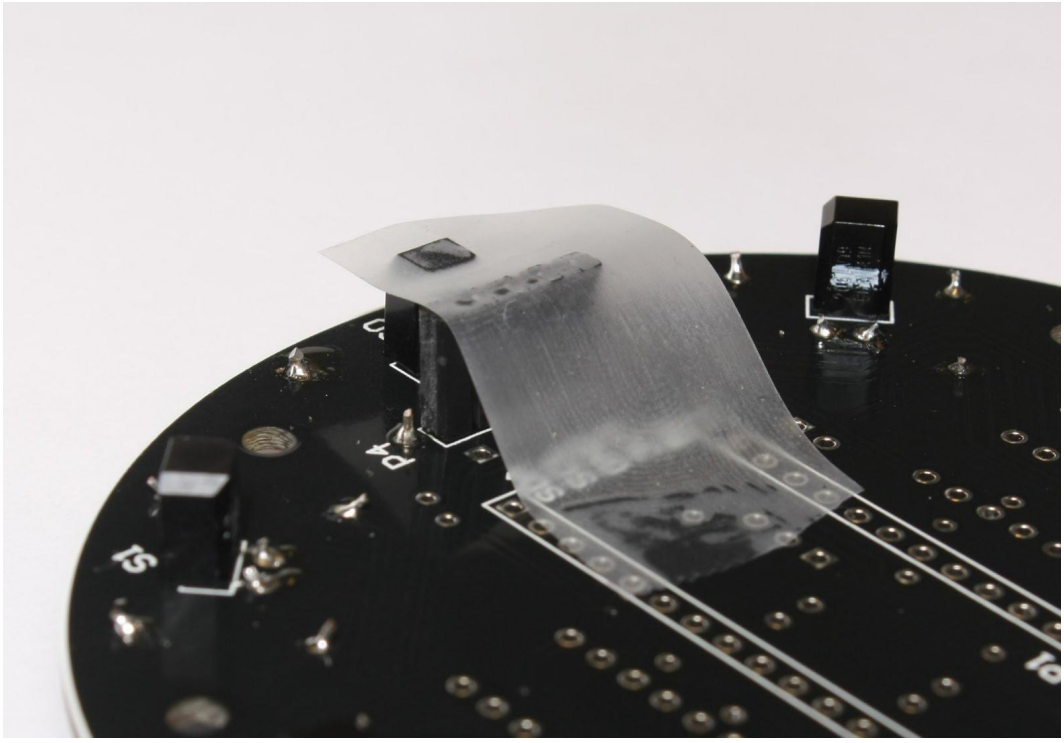
4. 5 pin headers

Headers are a way to connect things to a pcb. The Sensor board has several types of headers. The first we will solder in place is a 5 pin female header. If you don’t have a precut 5 pin female header, you can use the wire snips to break off five from a strip of breakaway headers.



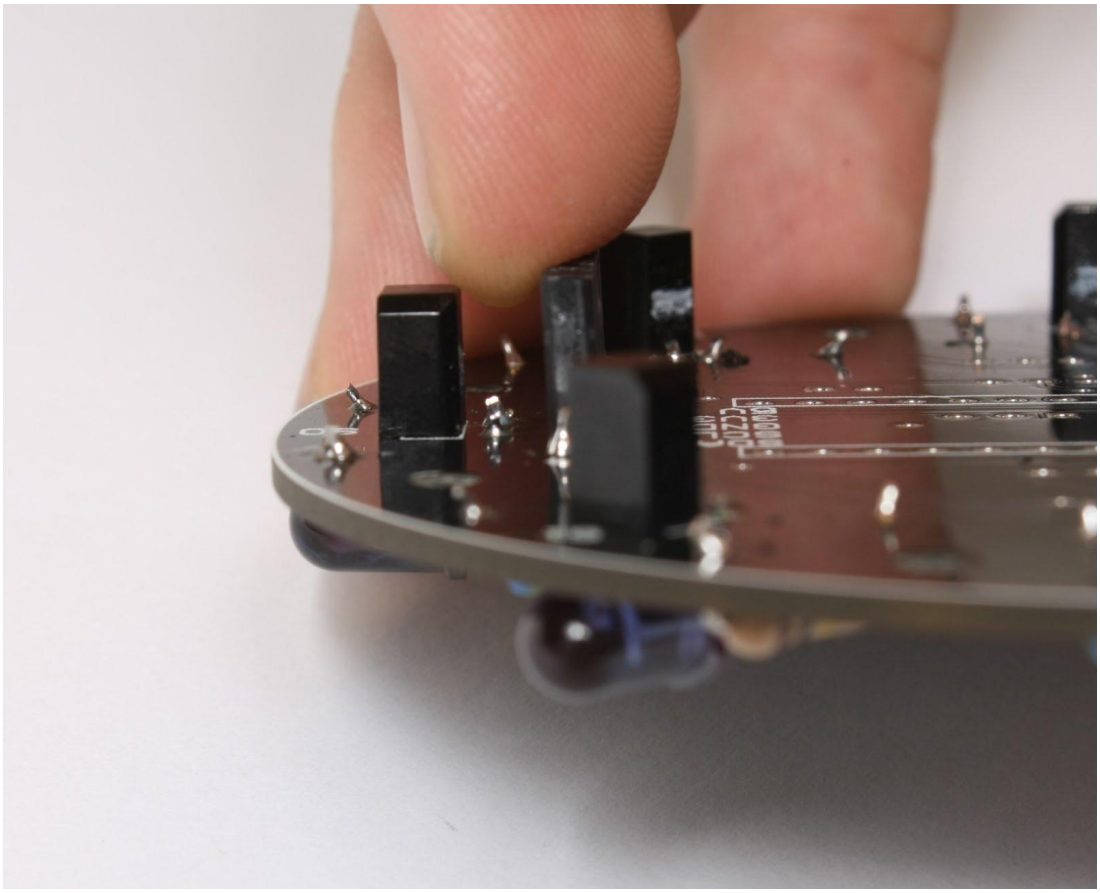
IMG18 five pin female header (broken of a breakaway header strip)

This header should be place on the bottom of the pcb on the slot marked P4.
To hold the header in place while you turn the pcb over you can use a piece of tape or, if you have it, blue tack works as well.



IMG19 use a piece of tape to hold the header in place

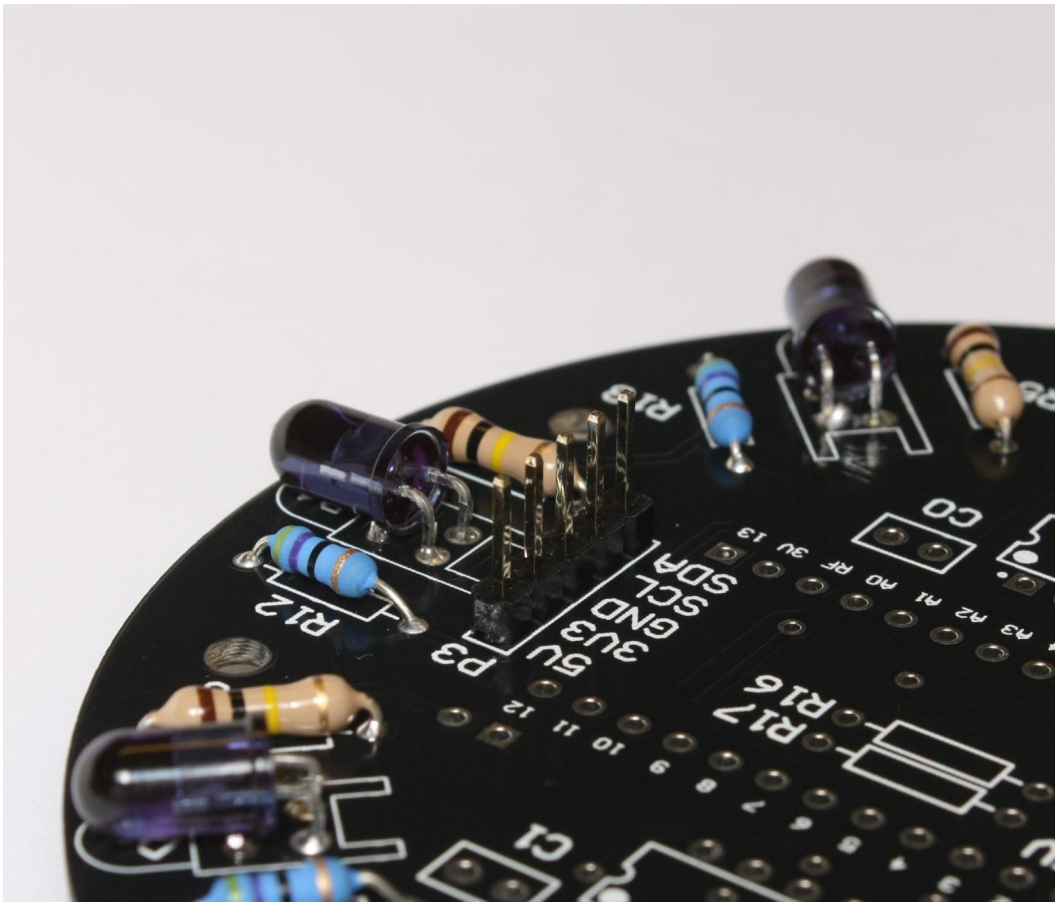
When you now turn the pcb over to the top side to solder, you start with soldering one of the edge pins first. After that has been soldered, turn the board over again and take of the tape. Now you can adjust the header to make sure it sits straight on the board.



IMG20 adjusting the header to make sure it sits straight.

When you are happy with the headers position, go ahead and solder the rest of the pins.

Now we have the 5 pin male headers, you will need two of them. They can also be snipped of breakaway headers. The procedure is exactly the same as for the female header, tape solder one pin, make sure it is straight and solder the other pins. The male headers should be placed on P2 and P3 on the top of the pcb.

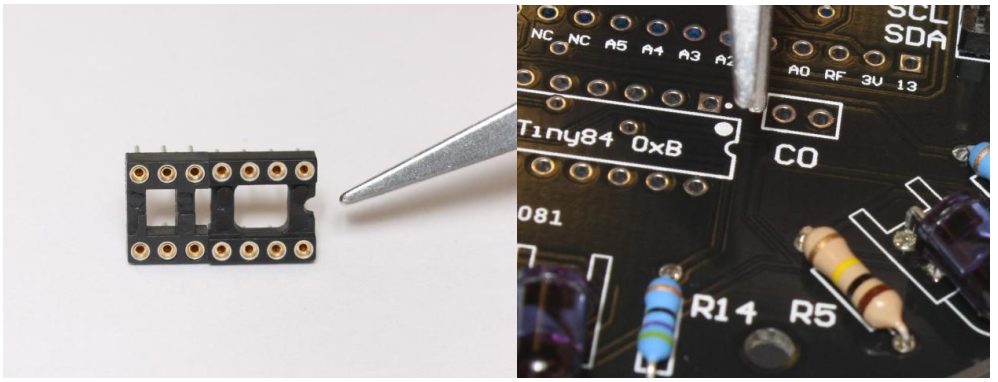


IMG21 5 pin male headers.

5. 14 pin IC sockets

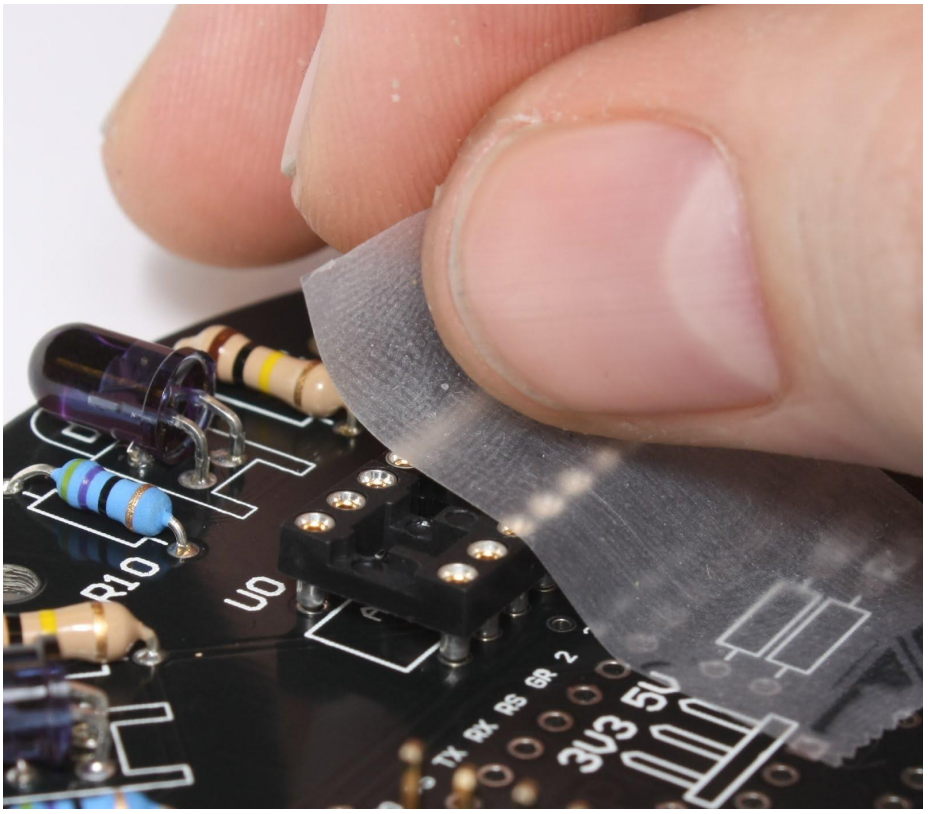
Now it's time to place the 14 pin IC sockets. These are used to hold the ATtiny84 ICs in place. The sockets have a small indentation on one of the shorter sides. You can find the same indentation on the drawings (called silkscreen) on the pcb. When placing the socket the indentations should be on the same side.

The sockets should be place in the slots marked U0 and U1.



IMG22 corresponding indentations on the silkscreen and the socket.

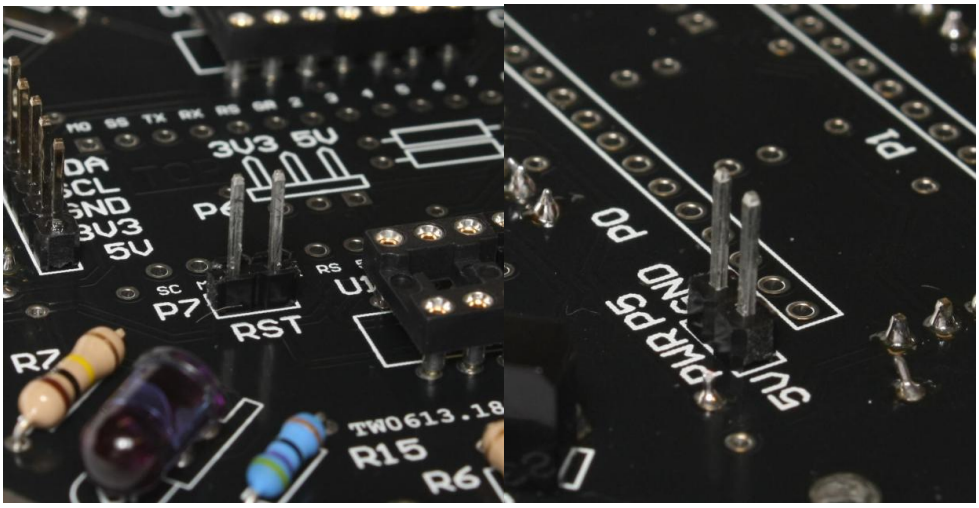
The trick with using a piece of tape to hold the socket in place should be used here as well.



IMG23 a piece of tape should hold the socket until you can solder it in place.

6. 2 pin male headers. Reset and Power

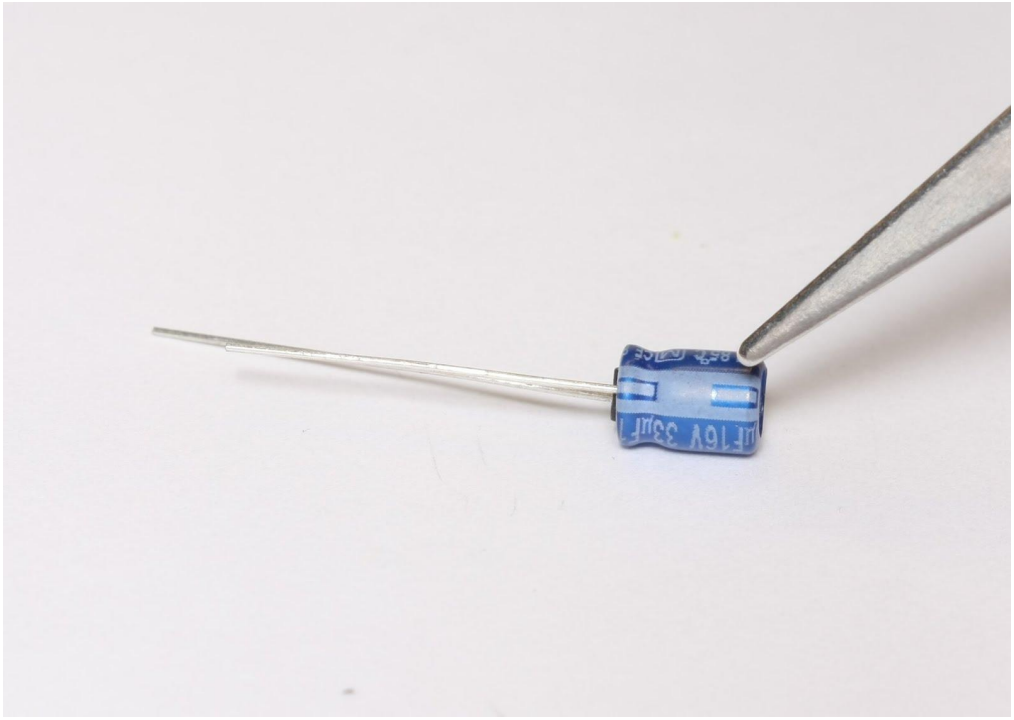
Two 2 pin male headers should be placed in the slots marked P7 on the top of the pcbn and P5 on the bottom of the pcb.



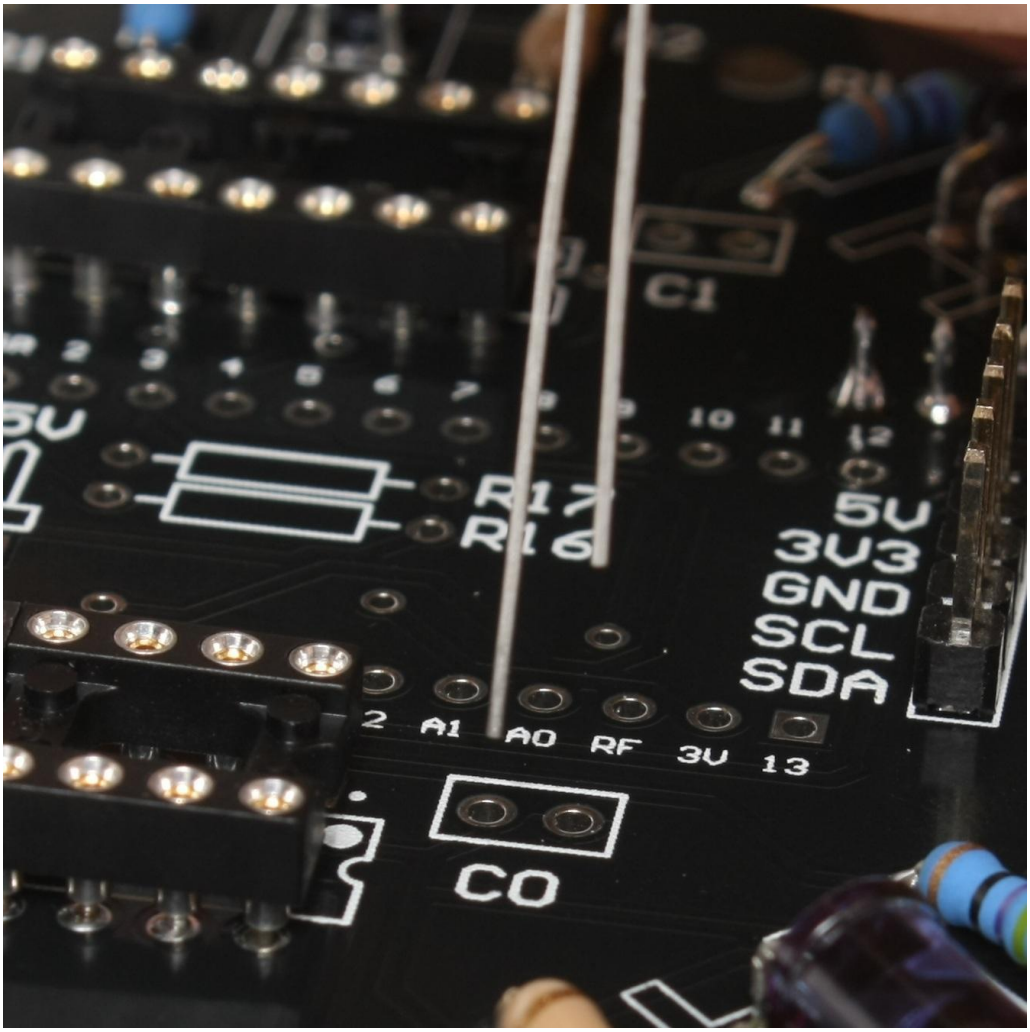
IMG24 two 2 pin male headers. one on the top of the pcb, the other on the bottom.

7. 33µF Capacitors

Two 33µF Capacitors have to be placed next to the IC sockets in the slots marked C0 and C1. These components, like the ir emitters, have to be placed in the correct orientation. One of the leads are shorter and denotes the negative lead. The line down the side of the capacitor also marks the side with the negative lead. This lead should be placed in the hole furthest from the sockets.

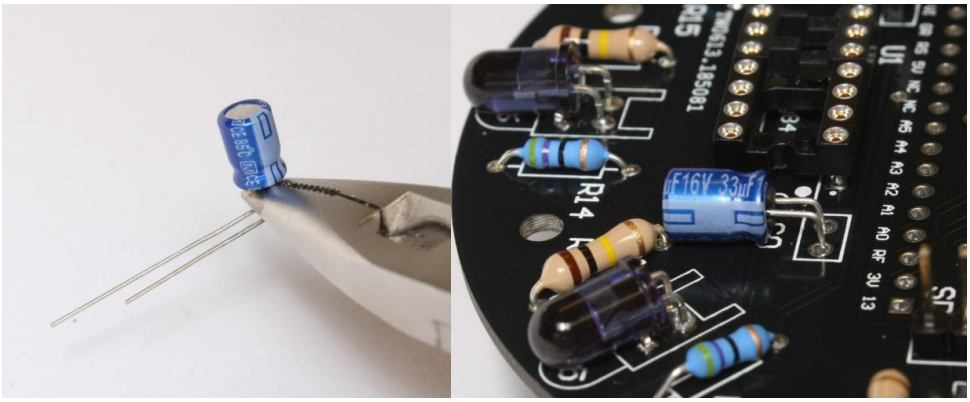


IMG25 The line down the side of the capacitor marks the side with the negative lead. This lead should be placed in the hole furthest from the sockets.

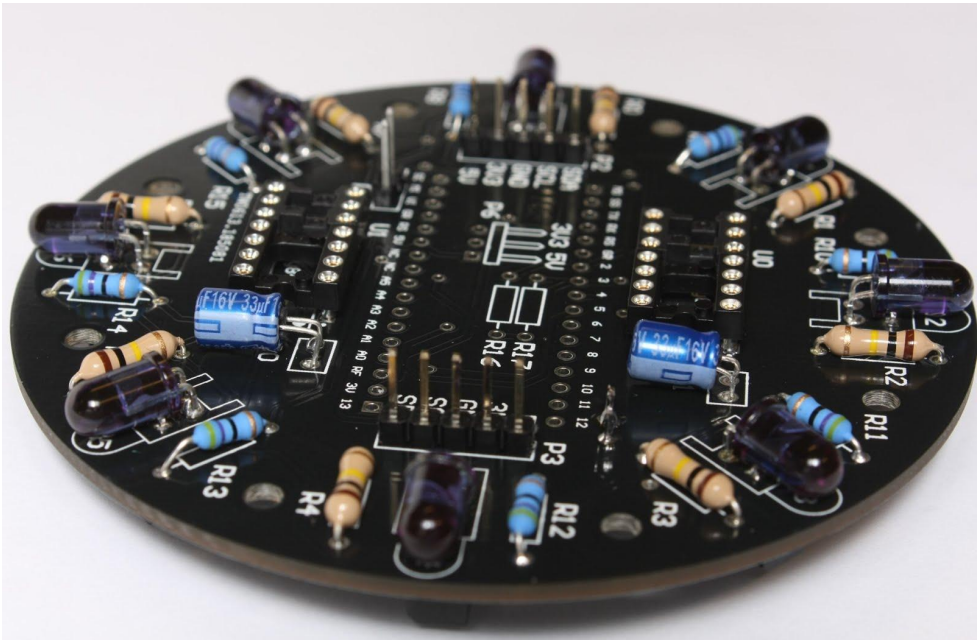


IMG25 negative lead in the hole furthest from the IC sockets.

This component also have to be bent in a similar fashion to the ir emitters. Make the bend so that the component itself covers the C0 and C1 text, with the line pointing away from the socket.



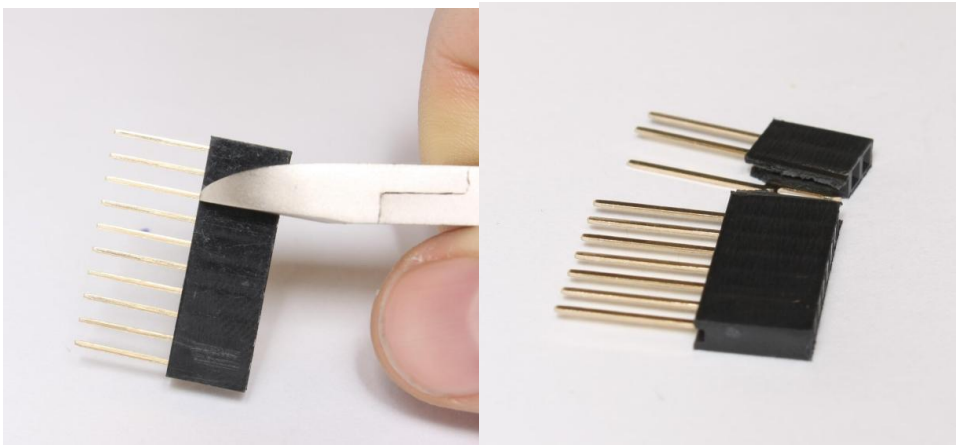
IMG26 bend and place the capacitor in the following fashion.



IMG27 Almost done now. Just a couple of components to go.

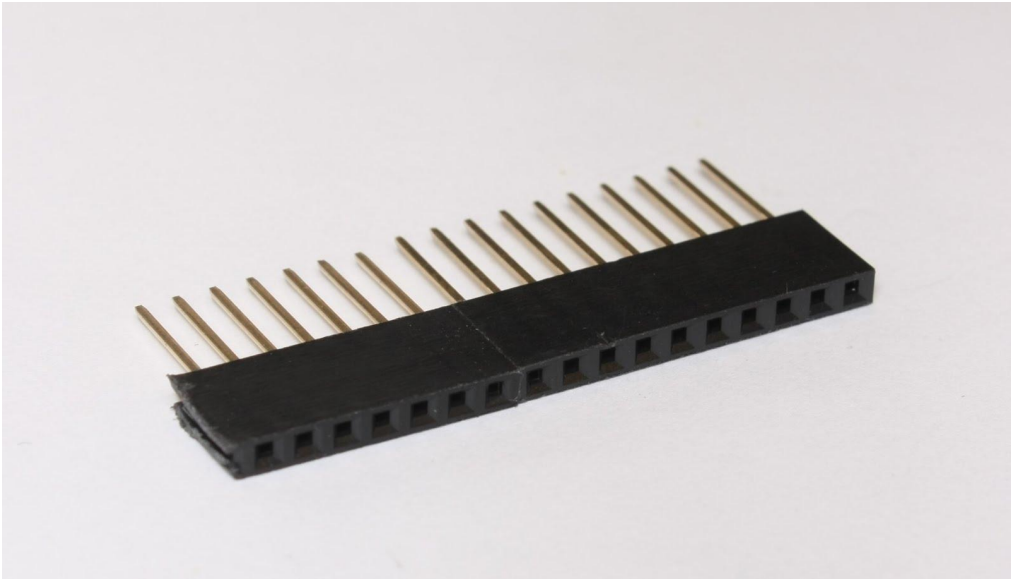
8. two 17 pin stackable headers (or four 10 pin stackable headers)

The last components to be soldered on are two 17 pin stackable headers. These headers look like the female headers we used earlier, but they have much longer legs. You may discover that you don't have 17 pin stackable headers. In that case you can make one out of two 10 pin stackable headers. To do this take one of the two 10 pin headers and place the wire snips in line with the third lead (see IMG28) and press hard. the third pin will break leaving you with one 7 pin header and one two pin header. put the two pin header away, you won't need it. You can use a sharp knife to clean up the break point if you want to.



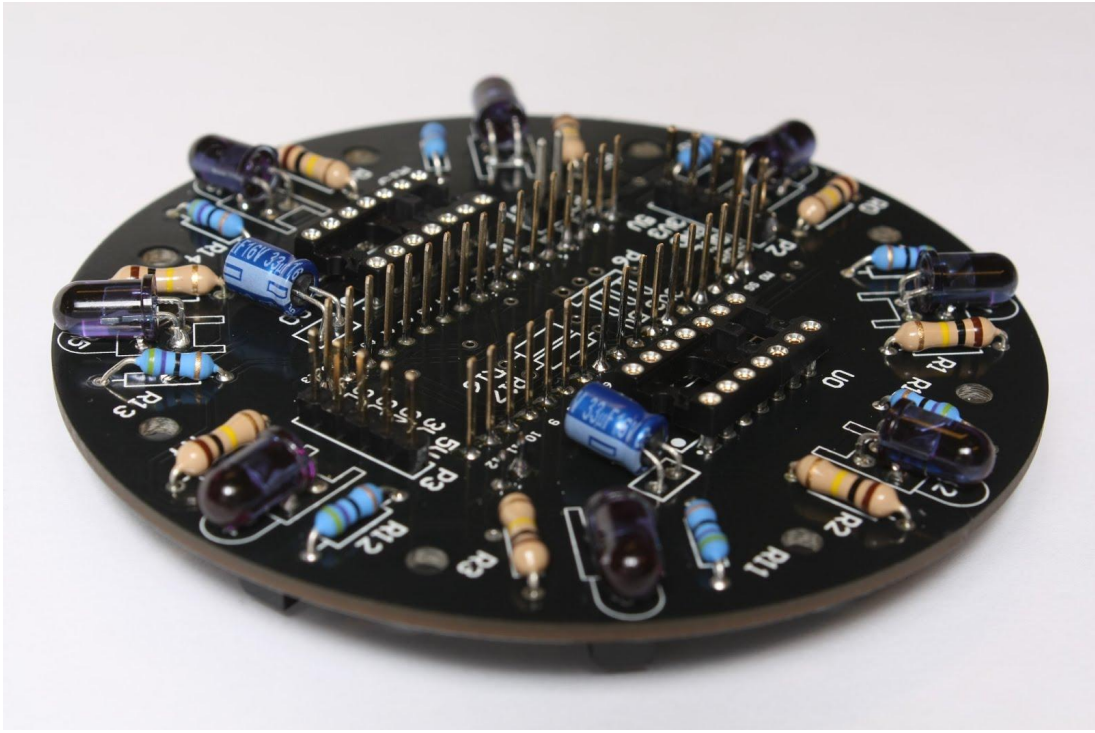
IMG28 Use the snips to make a 10 pin header into one 7 pin and one 2 pin header.

The new 7 pin header can be used in conjunction with another 10 pin header to make a 17 pin header.



IMG29 7+10 pin stackable header.

The 7+10 pin headers should press firmly into the slots marked P0 and P1 on the bottom of the pcb. If they don’t want to stick in place you can use the tape trick we used earlier to hold them down. Remember to first solder one pin, then adjust the header before soldering the rest of the pins. When the headers are soldered in place, the sensor board is complete.



IMG30 the completed sensor board!

Motor board PCB assembly

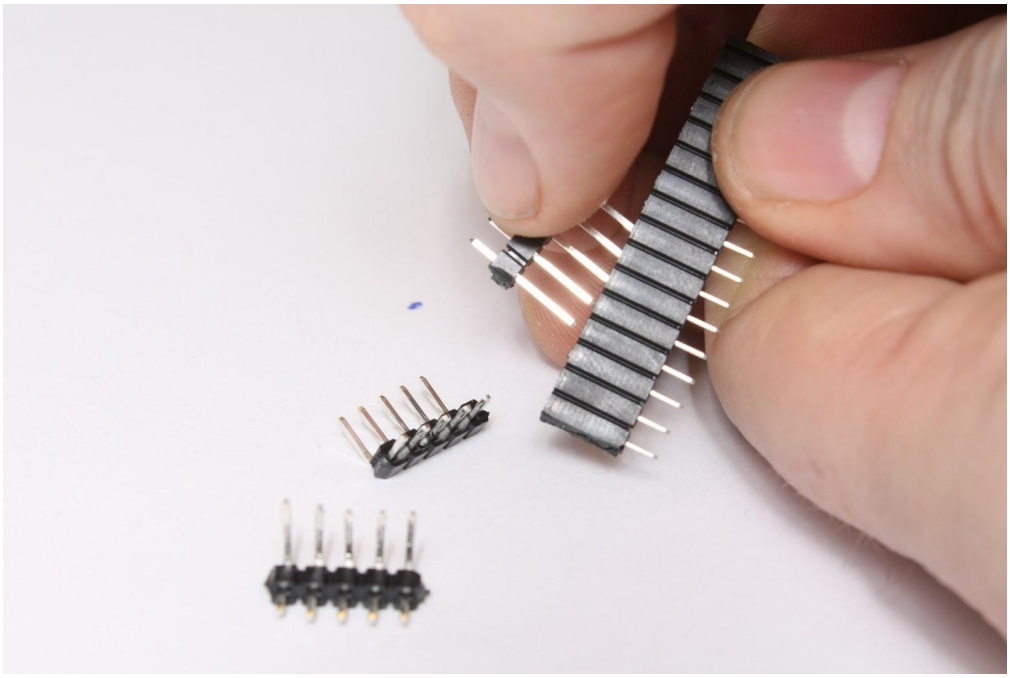
Tools required

- Wire snips
- Needle nosed pliers
- Soldering iron
- Solder

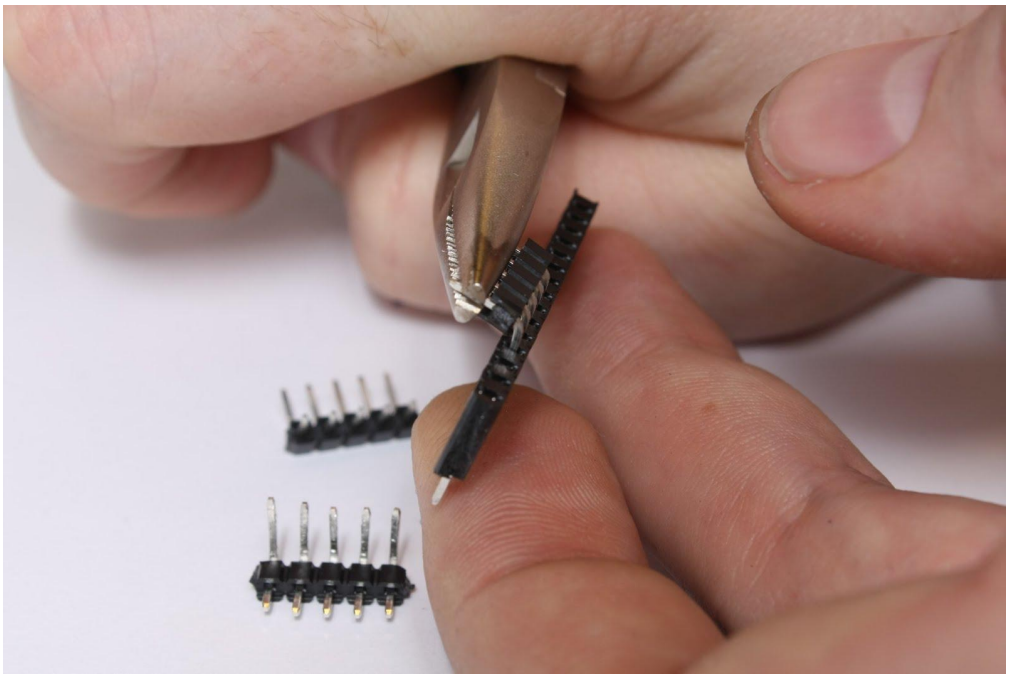
Assembly

1. 5 pin right angle male headers

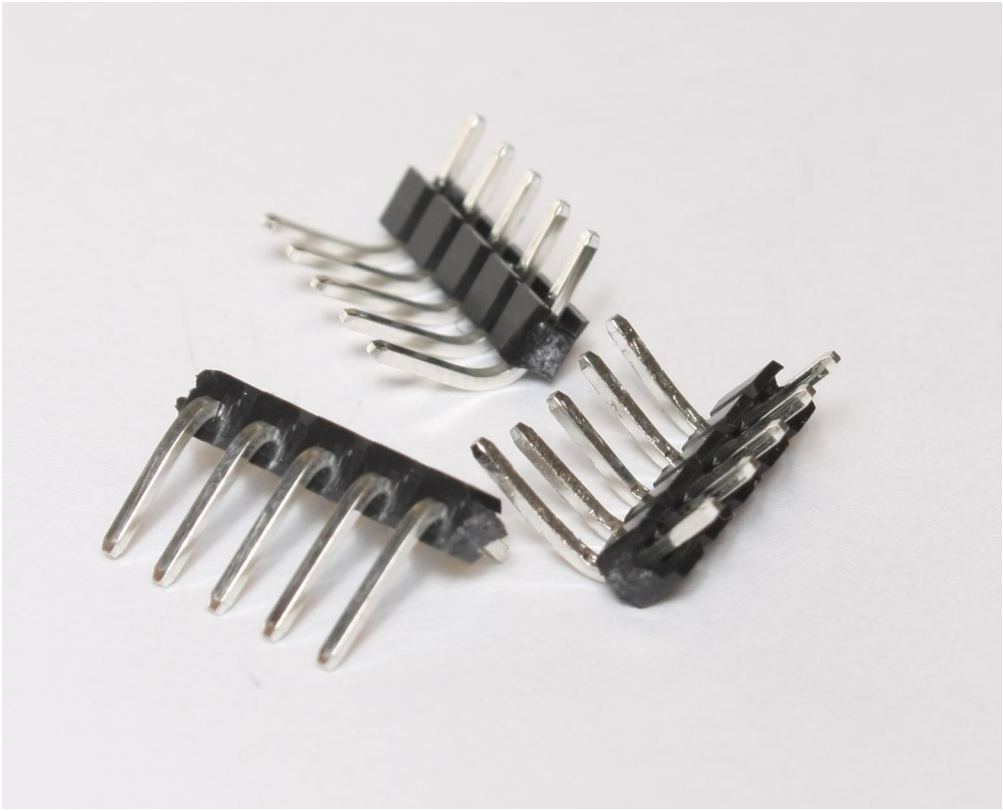
right angle male headers looks like regular male headers, but they have a 90° bend. If you don't have any male headers that look like that, I will show you how to make one out of a straight male header. Take the 5 pin male header and insert the long end into a row of female headers (see IMG31) and use pliers to bend the other part in at a 90° angle.



IMG31 start by inserting the long end into a female header

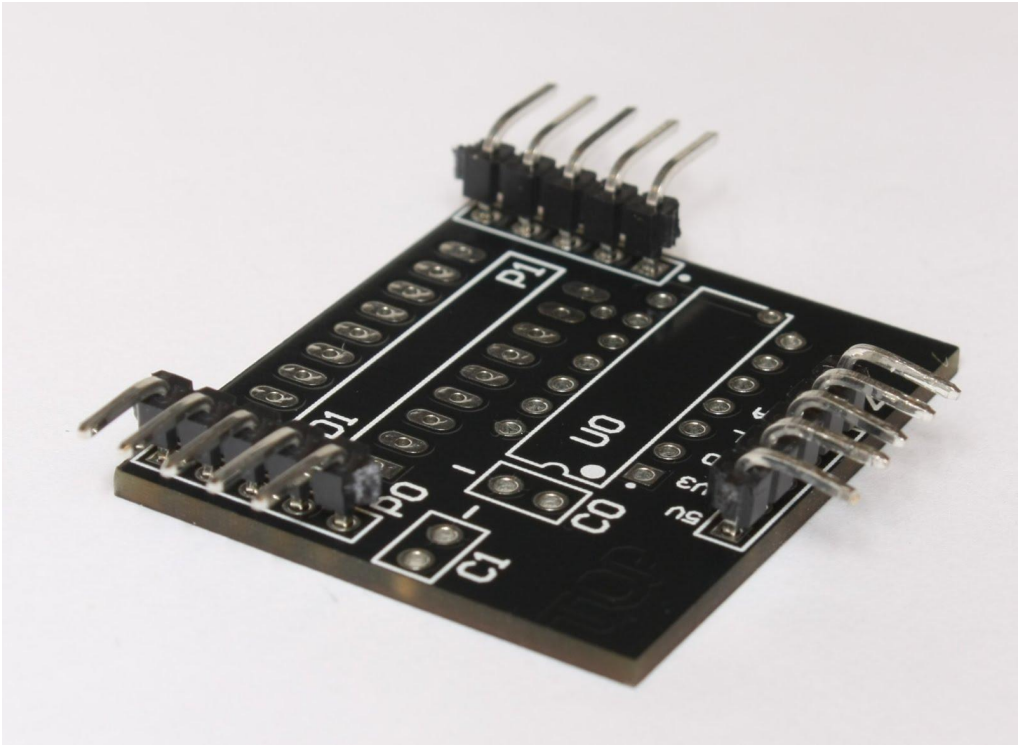


IMG32 bend the short end into a 90° angle.



IMG33 resulting three 5 pin 90° headers.

Next place these headers into the slots marked P0, P1 and P2 (see IMG34)

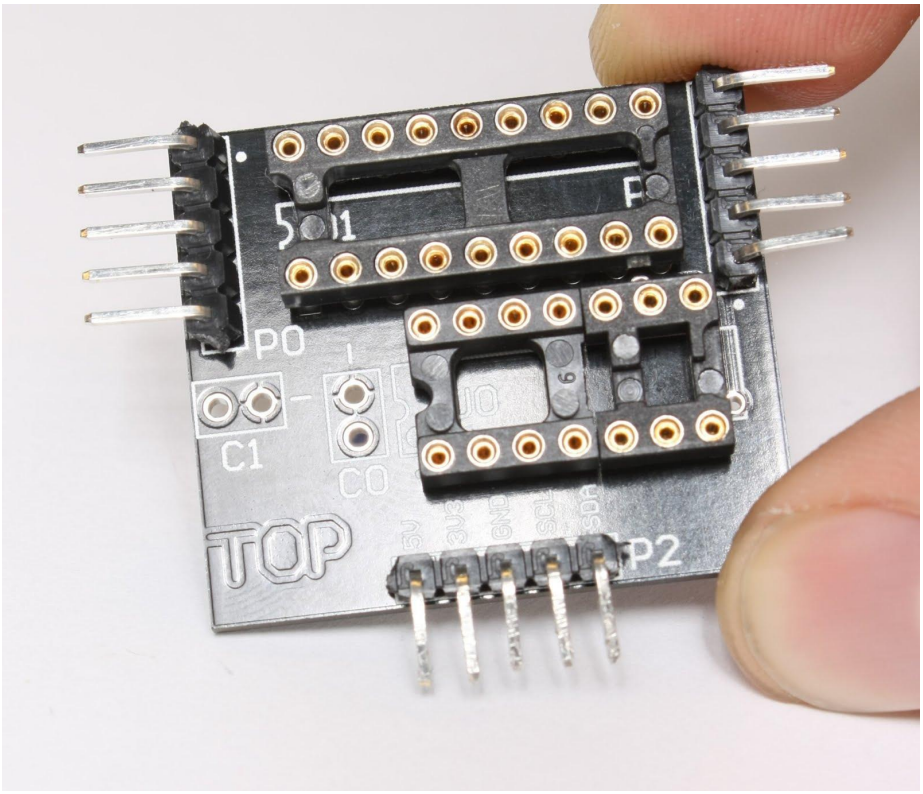


IMG34 headers in place.

Flip the pcb and solder. If they won't keep in place, you always have the tape trick.

2. 14 and 18 pin IC sockets

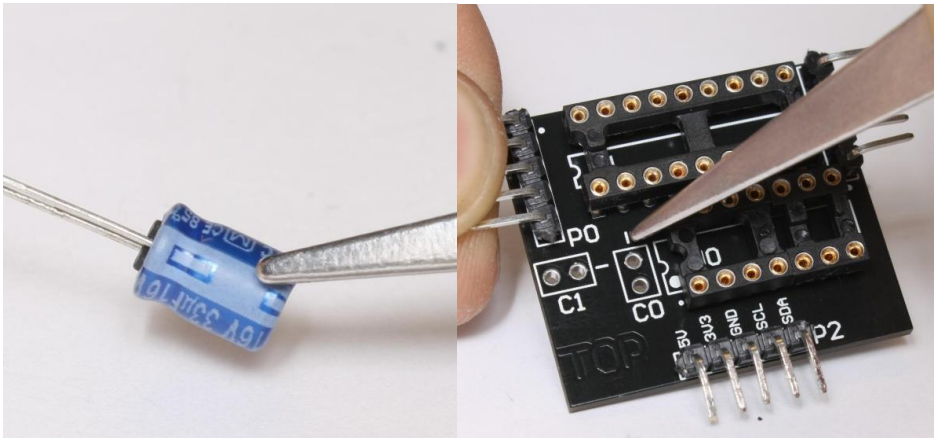
Next we have the sockets. You'll remember from the sensor board that the indentations on the sockets should line up with the markings on the silkscreen. Use the tape to hold them in place. The 14 pin socket goes in the U0 slot, and the 18 pin socket goes in the D1 slot.



IMG35 IC sockets in their respective slots.

3. 33µF Capacitor

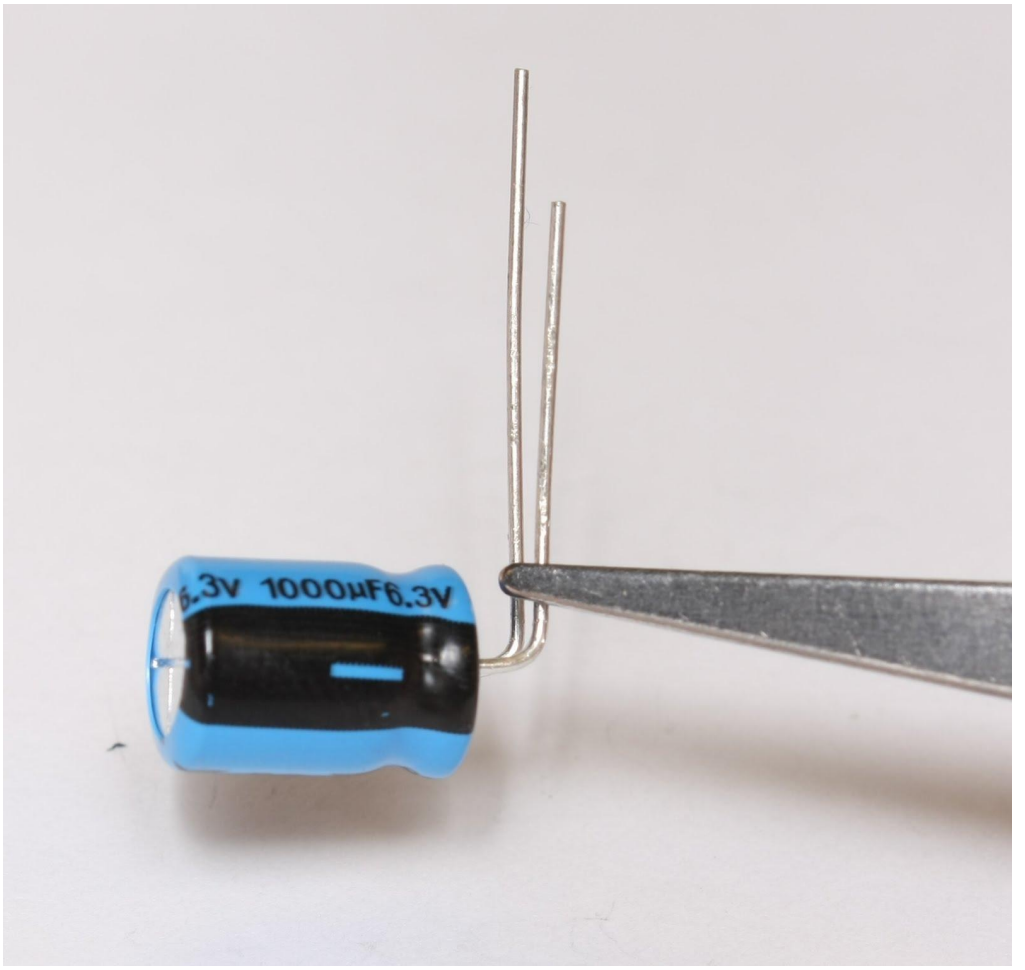
This board also has a 33µF capacitor that goes in the slot marked C0. This time the shorter negative lead marked with a line on the side of the capacitor, should be in the hole marked with a minus sign (see IMG36).



IMG36 the negative lead in the hole marked with a minus sign.

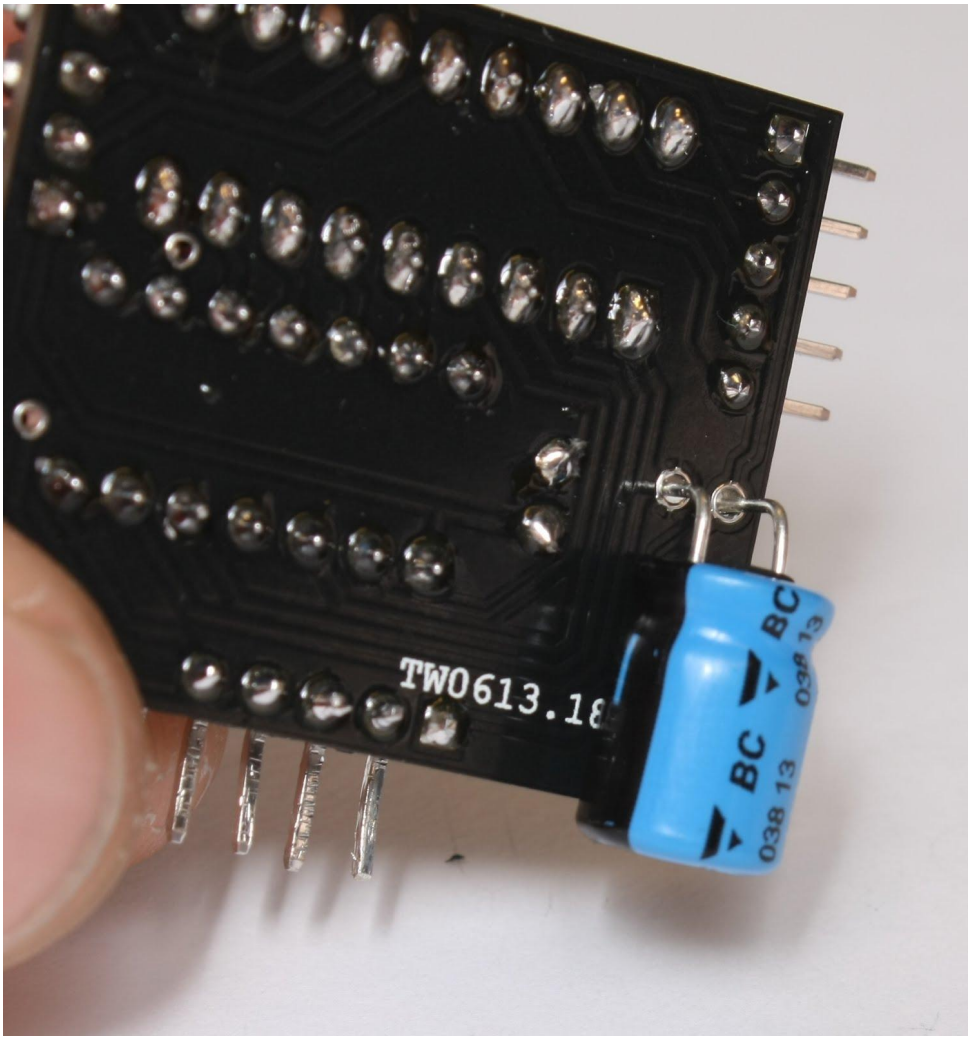
4. 1000µF Capacitor

This board also has a larger 1000µF Capacitor. The markings on this component are the same as the ones on the 33µF capacitor; the line marks the minus side. There is a twist here though, this component has to be mounted on the reverse side of the board from the silkscreen marked C1. It also has to be bent 90°. Let's start by bending it in the fashion shown in the picture IMG37, remembering to orient the black line in the same way as the photo.



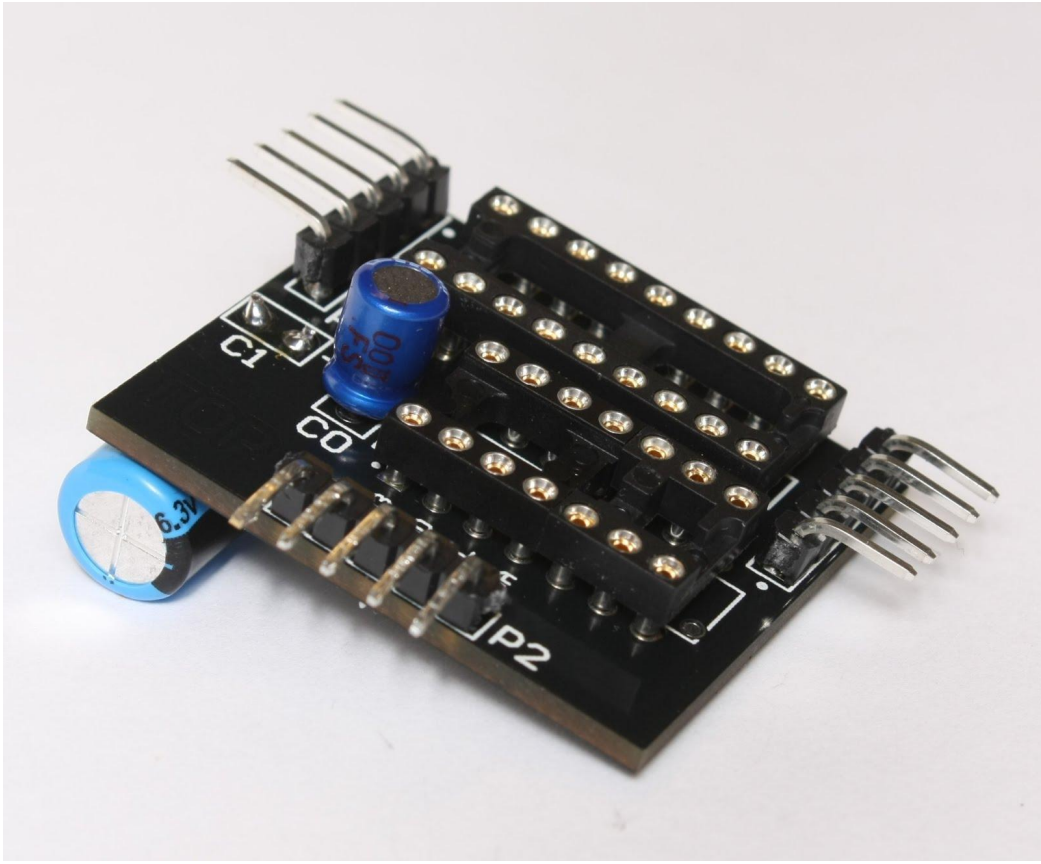
IMG37 bend the leads at a 90° angle the following way.

Now you have to insert the leads on the back of the board, reverse side of the silkscreen, with the black line pointing towards the center of the board (see IMG38).



IMG38 placement of the 1000µF capacitor.

The motor board is now complete!



IMG39 the completed motor board.

Robot assembly

Tools required

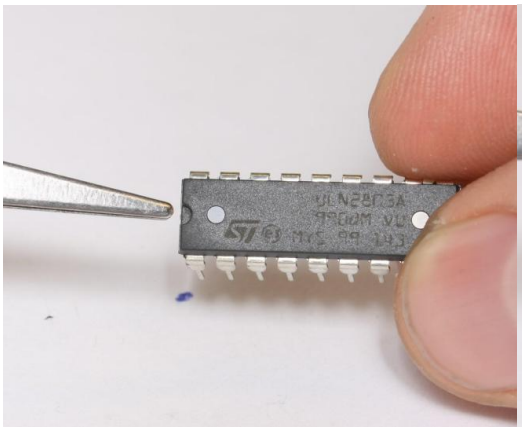
- Wire snips
- Needle nosed pliers
- Soldering iron
- Solder
- Flat head screwdriver (small and medium)

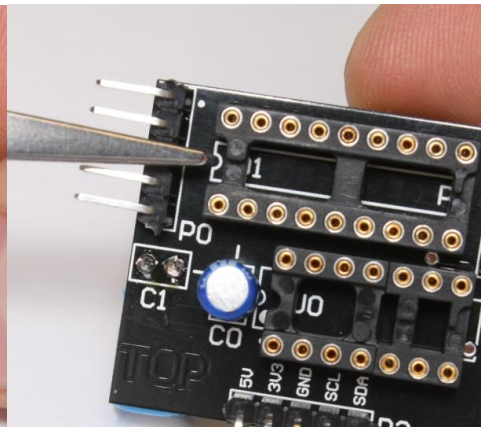
Assembly

1. Inserting the ICs

The sensor and motor boards uses four ICs. One 18 pin ULN2803 darlington driver and three 14 pin ATtiny84 microcontrollers. The ATtiny84 microcontrollers are all programmed in a separate way and you have to be careful not to mix them as there is no markings on the ICs that indicate which is is which. Let's start with the motor driver ICs.

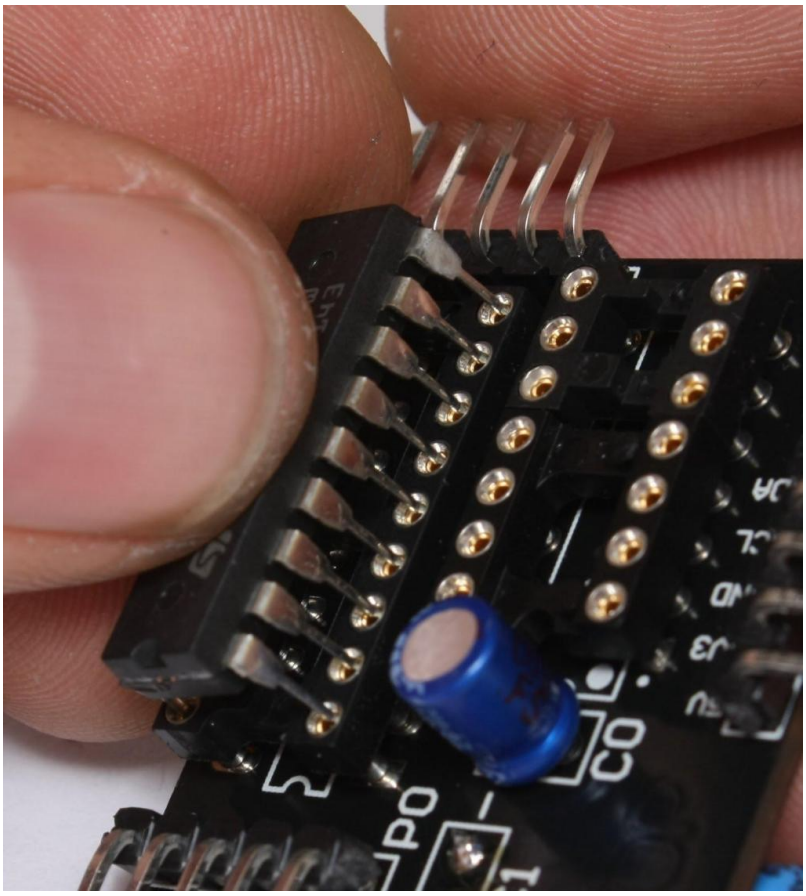
The 18 pin ULN2803 should be placed in the IC socket marked D1. There is only one IC socket with 18 pins, so that should be simple enough. The important thing is to make sure the orientation of the IC is correct. Similar to the sockets, the ICs also have a small indentation on one of the short sides. This indentation should line up with the indentation on the socket and the silkscreen (see IMG40). This is the same for all of the ICs.





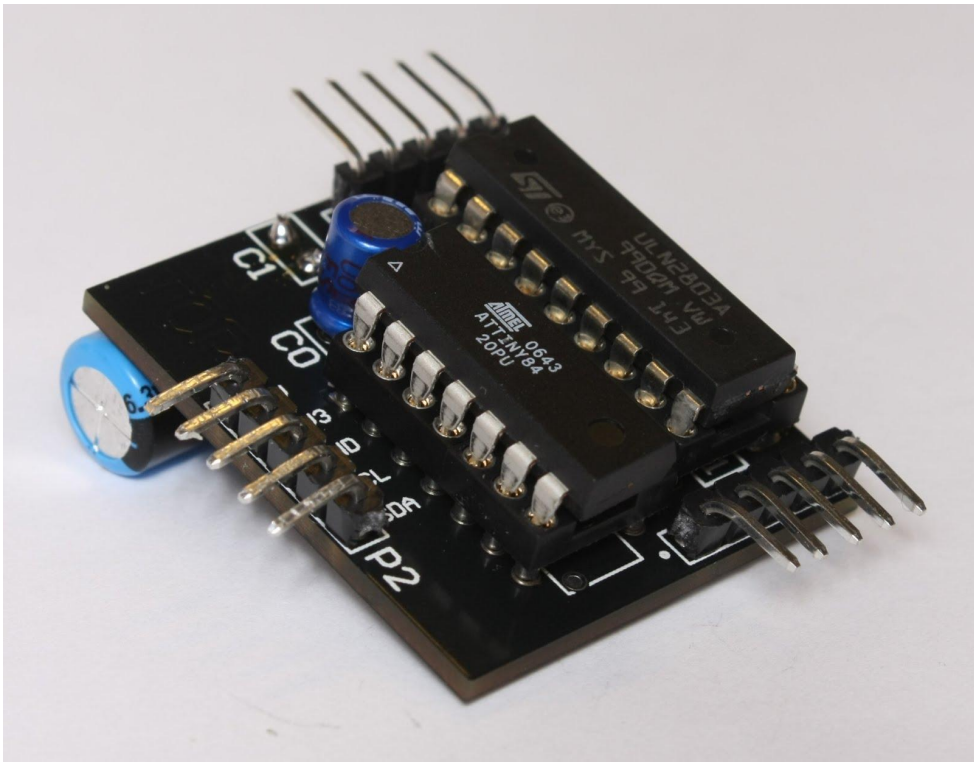
IMG40 Indentations on the same side.

Insert the IC by aligning all of the pins with all of the holes in the socket. once aligned the IC should go in with a gentle squeeze. Don’t force the IC into the socket. Once the IC has entered the socket you can squeeze firmly to secure it in place.



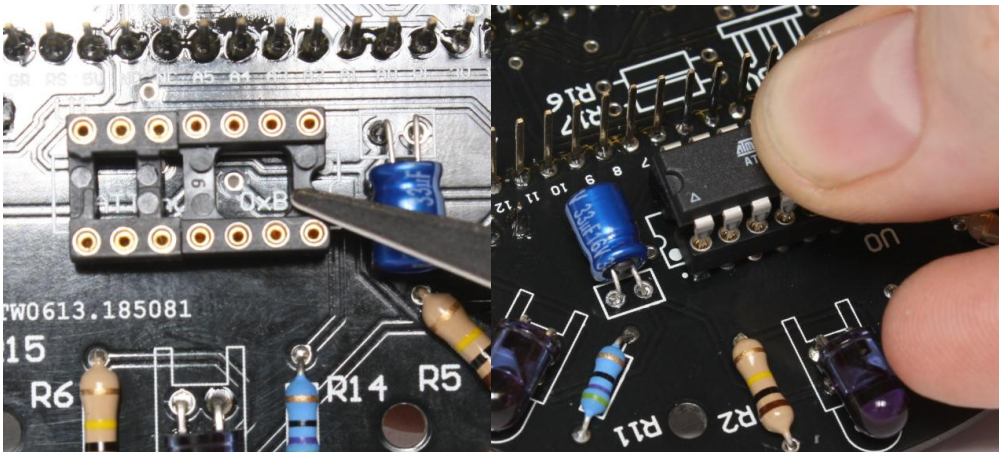
IMG41 align holes and squeeze. Don’t force it in. Secure with a firm squeeze once the IC has entered the socket.

The ATtiny84 should be placed in the slot marked U0. This is the only socket left, so this should be simple enough. *Make sure you pick a ATtiny84 that is programed as a motor driver.* Align indentations, squeeze into the socket.



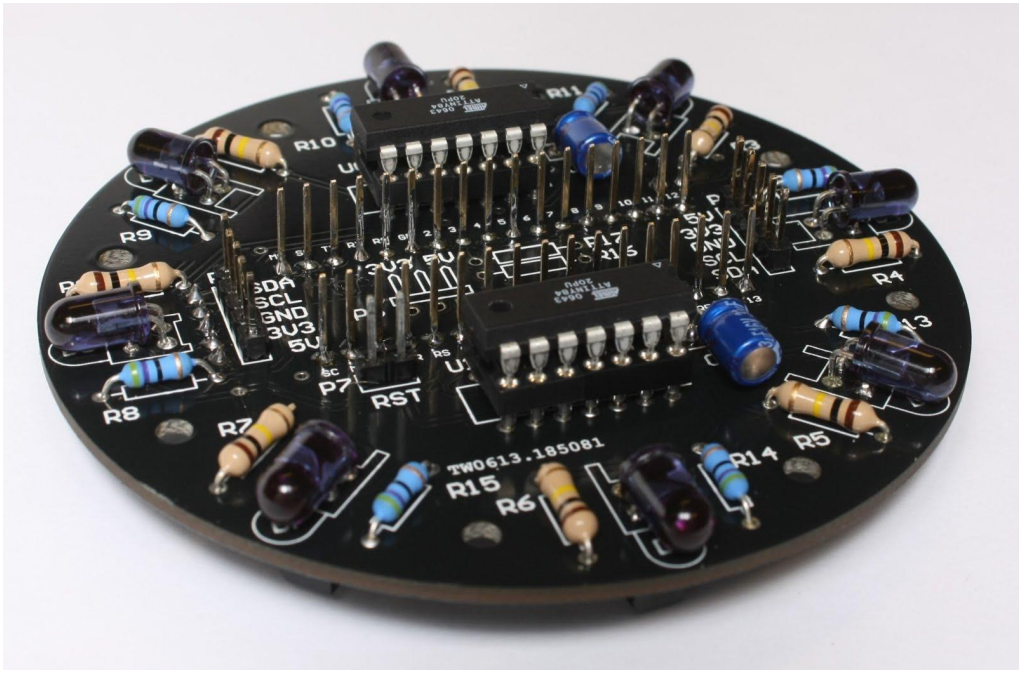
IMG42 Fully assembled motor driver board.

Now we have to insert the ATtiny84 ICs on the sensor board. You do this in the same way, but make sure you don't mix the two differently programed ICs. The front sensor IC goes in the U0 socket and the back sensor IC goes in the U1 socket. You can also tell them apart by the 0xF marking under the front socket and the 0xB under the back socket (This is the I²C address for the chip, if you were wondering).



IMG43 Make sure the correct chip is in the correct socket, and squeeze them in.

Congratulations, the sensor board is now fully assembled.

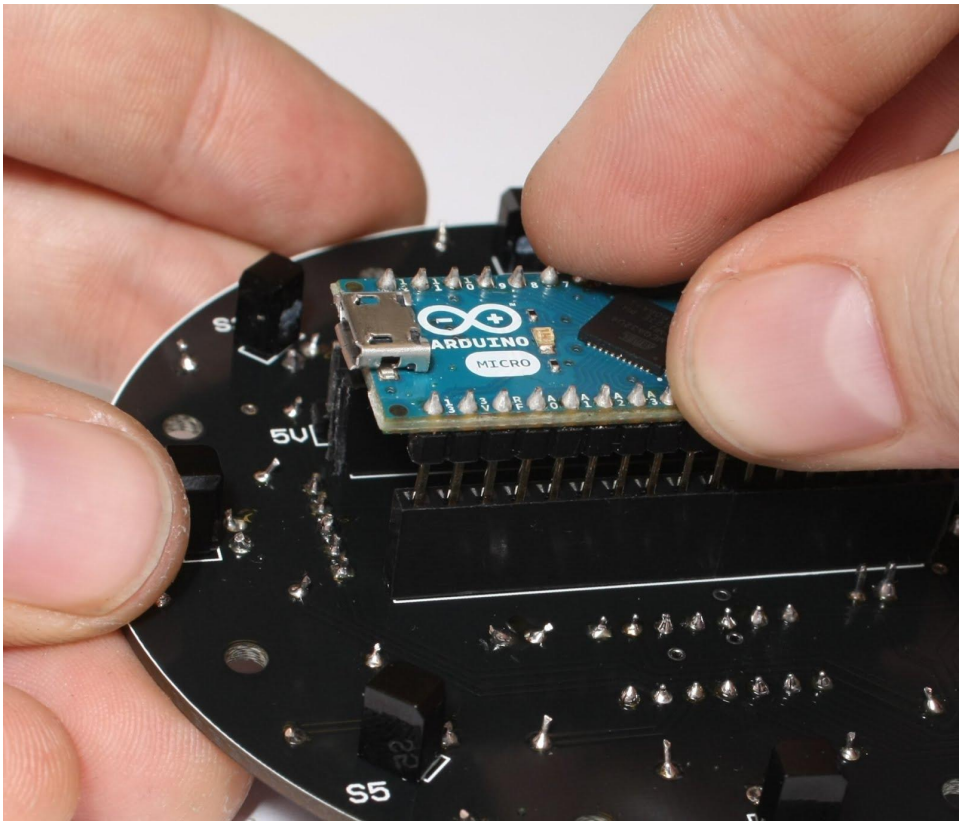


IMG44 Fully assembled sensor board.

2. Connecting the PCBs

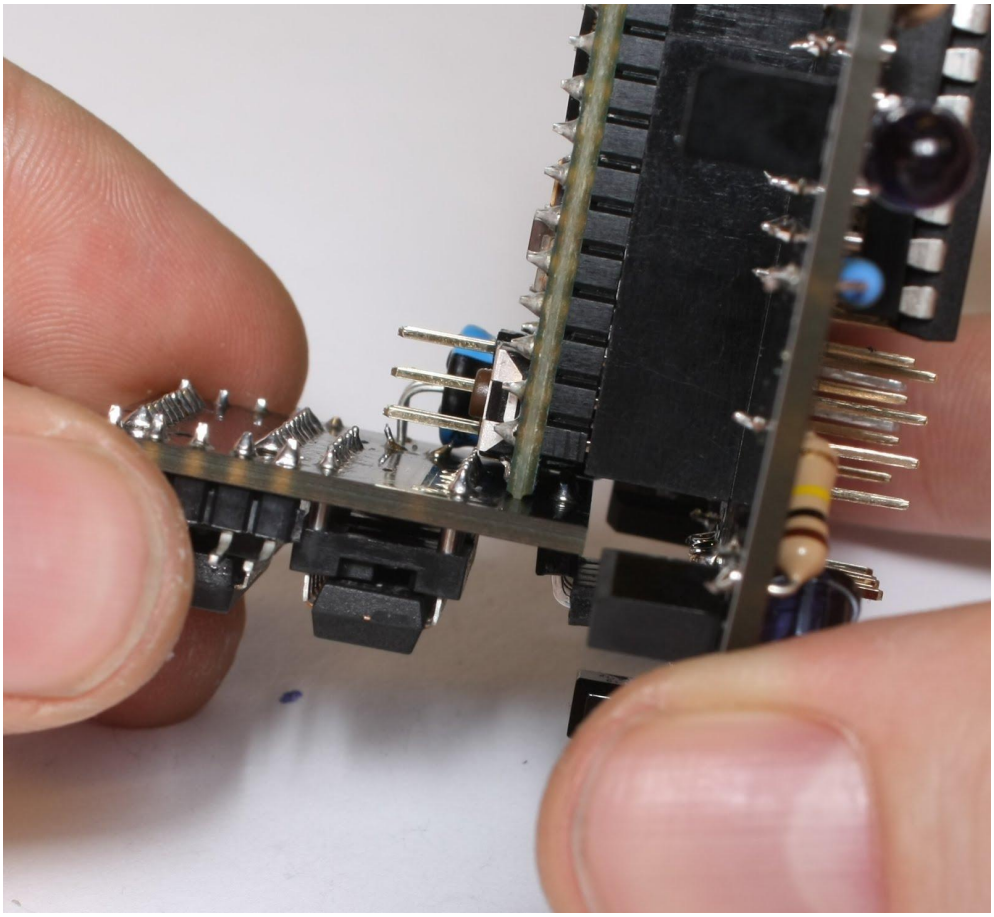
Now we will connect together the PCBs. You'll need the motor driver PCB and the sensor board we made earlier plus the Arduino Micro board.

First, start by connecting the arduino board to the sensor board. The male headers on the arduino go into the two rows of the female headers on the bottom of the sensor board (see IMG45) the micro USB jack on the arduino should point towards the IR receiver marked S4.



IMG45 insert the arduino into the bottom of the sensor board with the USB jack towards the sensor marked S4. This side will be the back of the robot.

Next you'll need to connect the motor board to the sensor board. Plug the male headers on the motor board marked P2 into the female headers marked P4 on the bottom of the sensor board. It should be a tight squeeze to get the solder points of the motor board passed the edge of the arduino board. You may have to carefully bend out the motor board to make it go in all the way. Once in place, the assembly should look like it does in IMG46.



IMG46 fully inserted motor board.

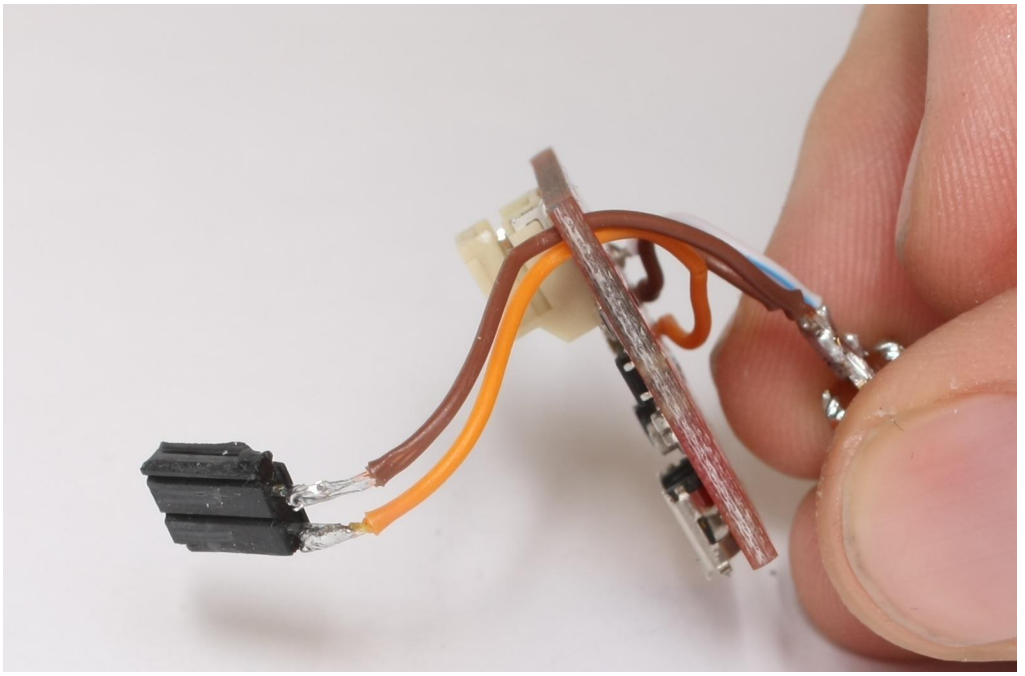
3. Making the power switch assembly

Now we need to use the soldering iron again. By now I will presume that you know a bit about how to solder. The new thing now is that we are going to solder wires. Start by cutting two wires about 1 cm long. Strip of about 3 mm of the insulation on the end of the wires. Now touch the soldering iron to the bare copper wires while touching it with some solder on the other side. This is called tinning the wires and makes them easier to solder to other thing later. The solder coating on the wires should be thin, or else you won't be able to feed the wires through the holes on the PCB. Now that you have tinned the wires you can put one of the solder covered ends of the two wires through the holes marked EN and GND on the bottom (side without the USB jack) of the red powercell PCB, one wire in each hole. Solder them in place on the reverse side.

Now solder the other ends of the cables to the switch in the way shown on in IMG47 (the two short wires).

IMG47 switch and powercell assembly.

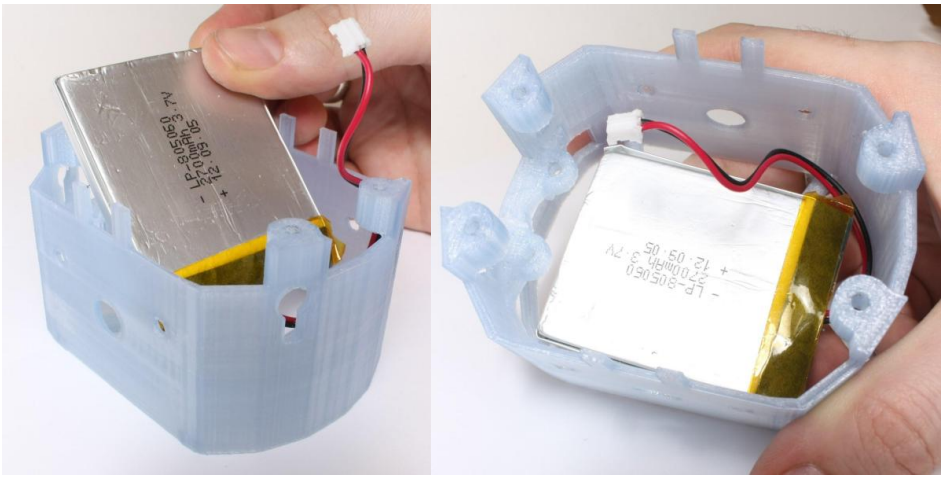
Now cut to longer pieces of wire, one red and the other black/brown, about 2 cm long and tin them like we did the others. Solder the red one to the VCC hole and poke it through the back mounting hole like shown in IMG47. The other wire is soldered to the last terminal on the switch and poked through the same hole. The two longer wires ends are soldered to a 2 pin female header (see IMG48).



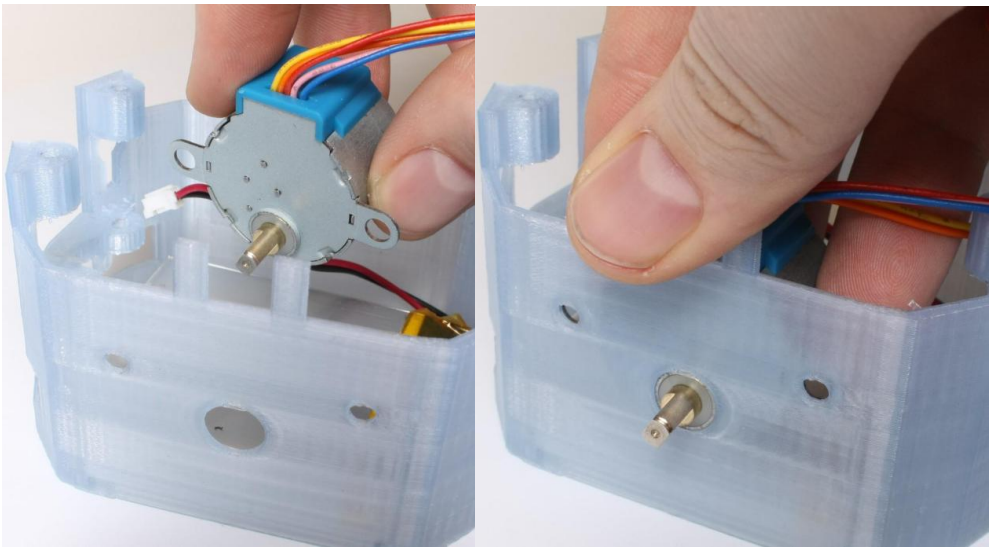
IMG48 the last two wires are soldered to a 2 pin female header.

4. Final assembly

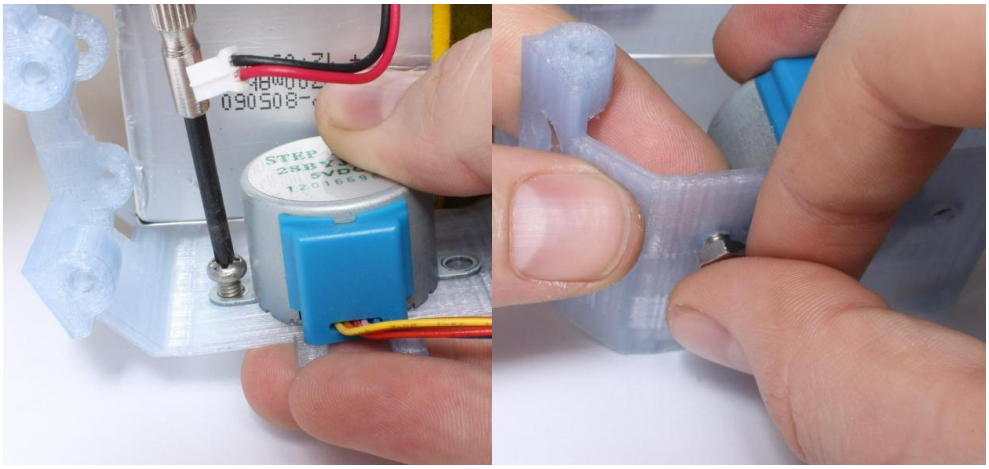
Now we are almost done. Take an empty chassis and put a battery inside. the wires should point towards the front of the robot, the short end of the chassis without the gaps.


IMG49 Battery placement

Now take one of the motors and poke the shaft through the larger middle holes on the long part of the side of the chassis.


IMG50 poke the shaft through the big hole.

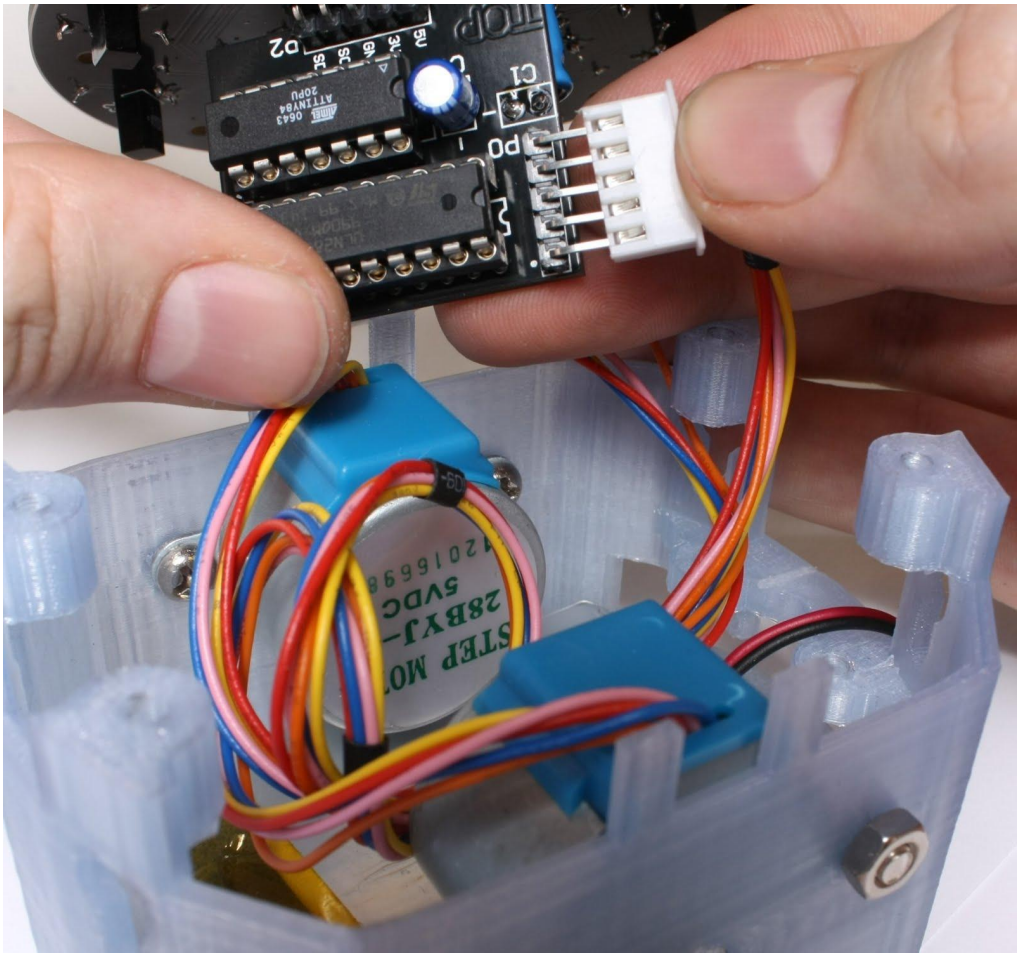
Align the mounting holes on the motor with the smaller holes on the side of the robot and secure with a screw from the inside of the chassis and a nut on the outside.



IMG51 secure with a screw and a nut through both mounting holes on the motor.

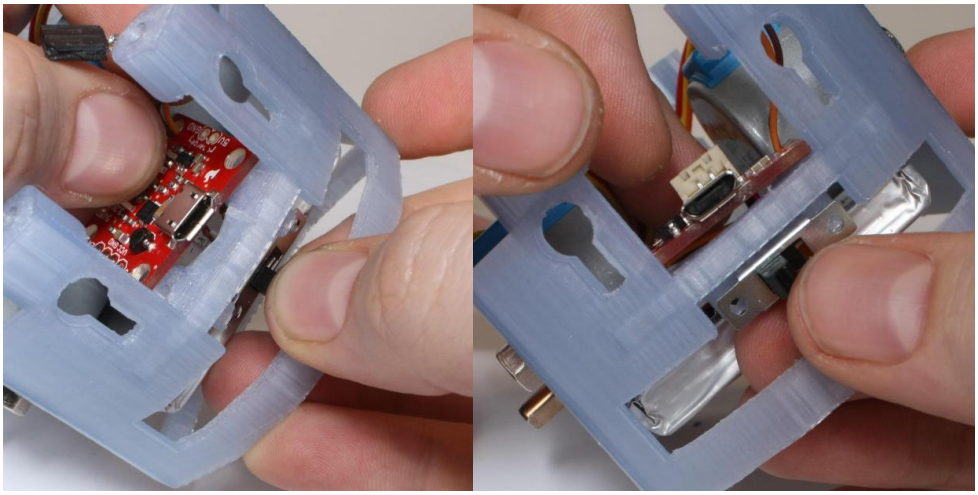
Now you can mount the other motor in the same way.

Try to twist the cables on the motors so they both coil up nicely. Attach the left motor plug (remember that the back of the robot is the short side with the gaps and holes) to the male headers on the motor board marked P0. The plug should be inserted with the red wire on the side of the header that has the white dot. After the left motor is attached you can attach the right motor to the last header on the motor board. (Remember that the red wire goes to on the side with the dot).



IMG52 Attach the left motor plug to the male headers on the motor board marked P0.

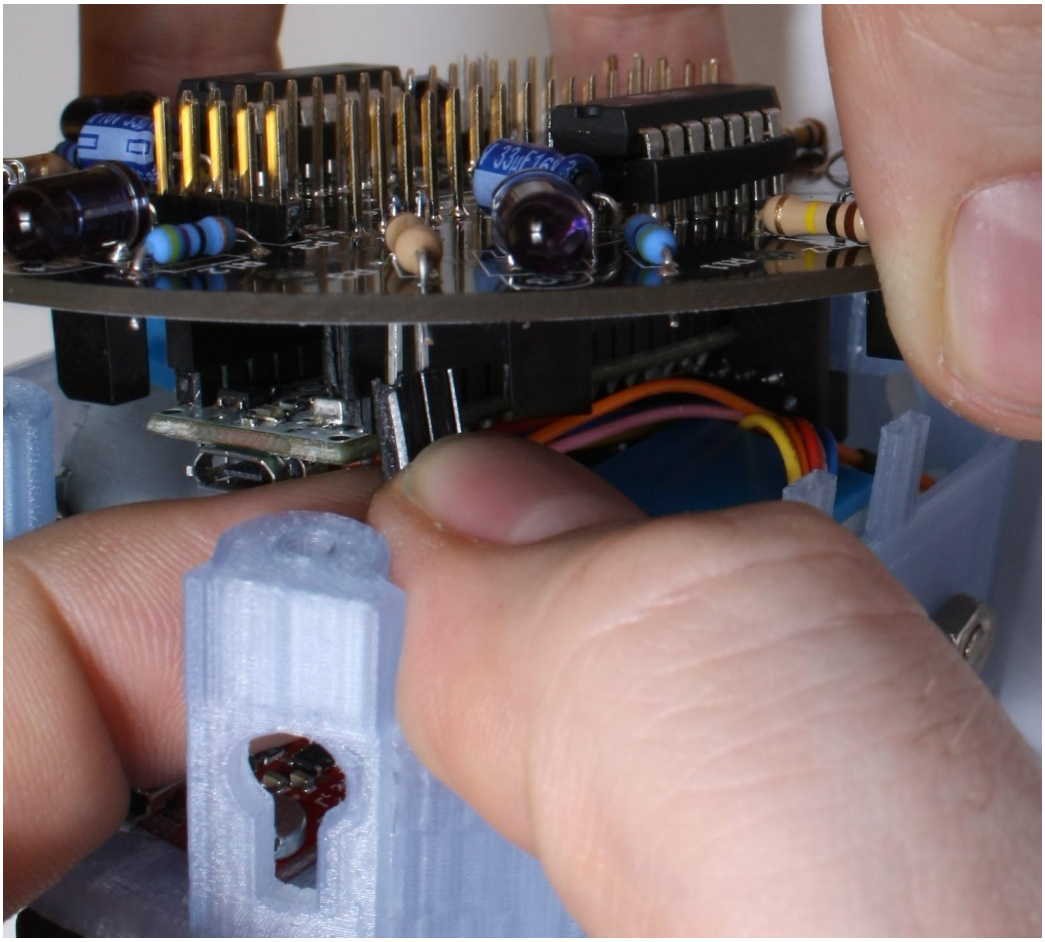
Now it is time to connect up the power. Start by plugging in the battery leads to the red power cell board. The connector on the battery only goes in one way so you don't have to worry about plugging it in the wrong way. After you have plugged in the battery it's time to secure the power switch. The best way to describe this step is to look at the image IMG53.



IMG53 Separate the powercell board and the switch and position like this.

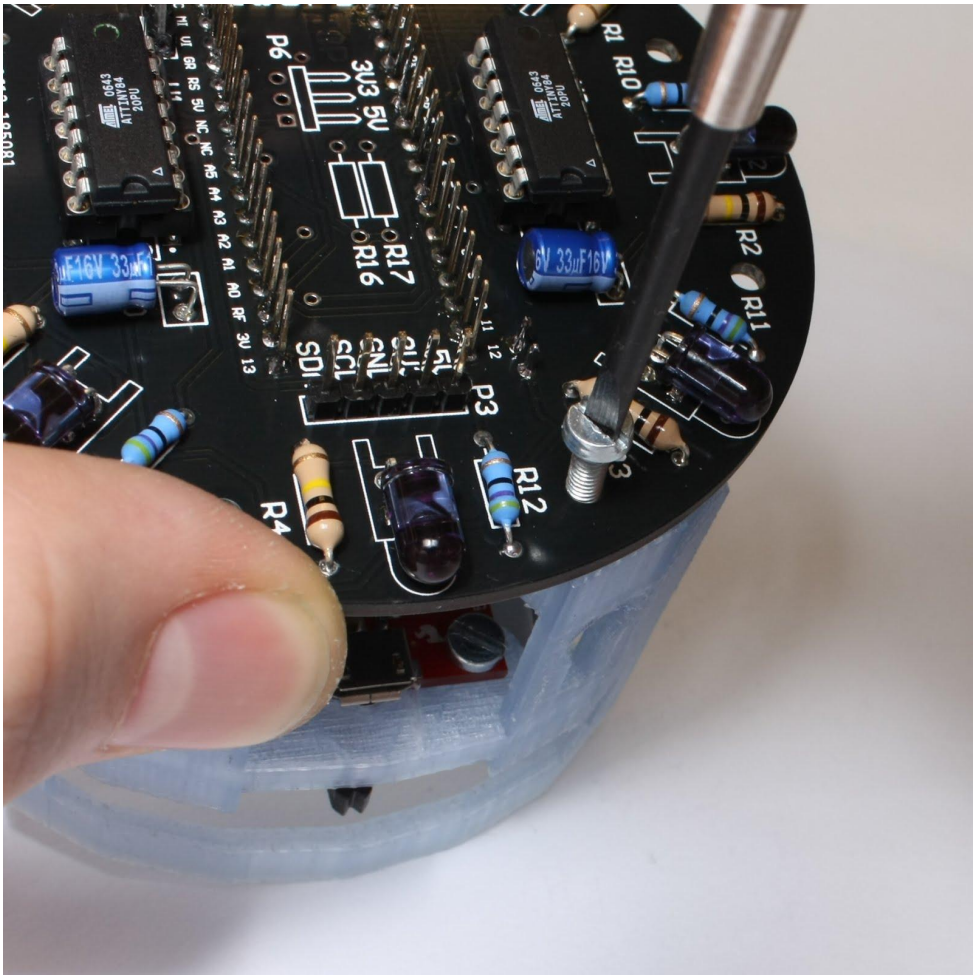
Secure the switch with two 5mm M2 screws and the powercell board with two 5mm M3 screws.

While tucking the motor board in to the front of the robot, take the female header from the powerCell board and connect it to male header marked PWR P5 on the bottom of the sensor board, the red wire side goes to the 5V pin, and the other to the GND marked pin.



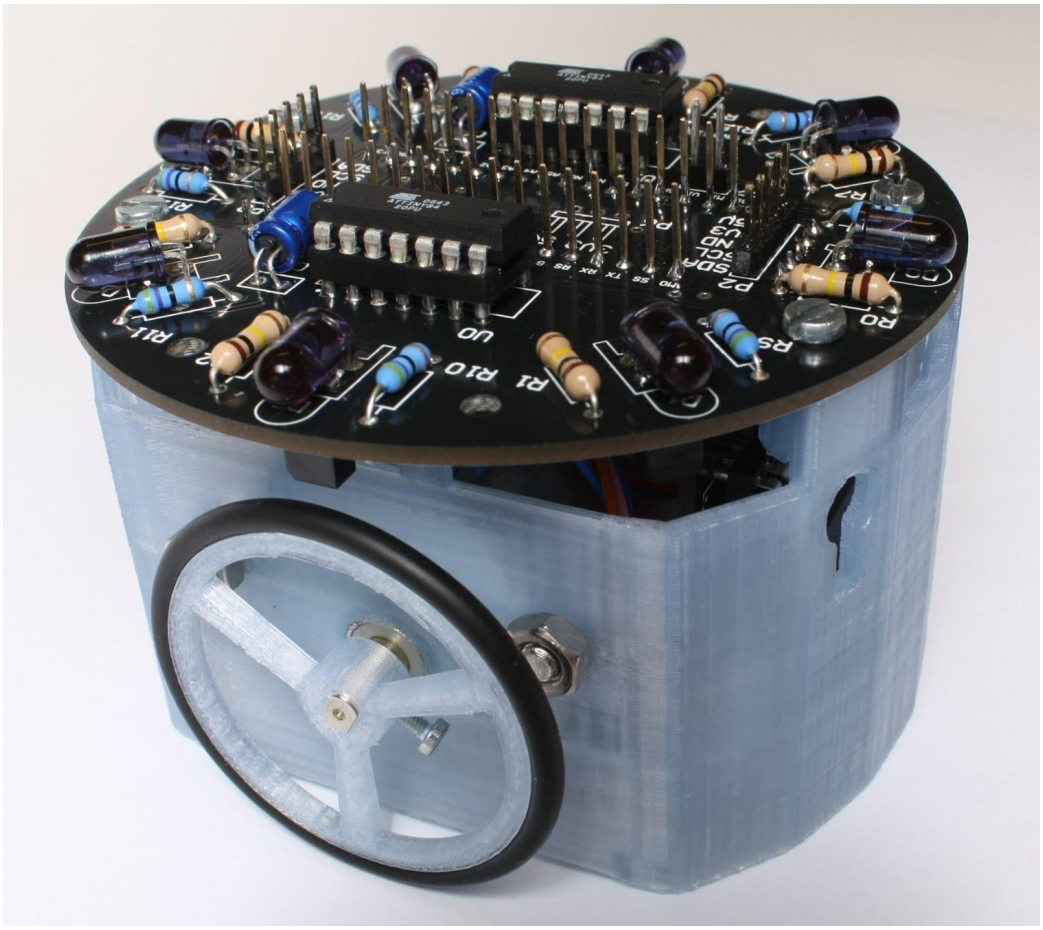
IMG54 Remember that the red wire goes to 5V and the other to GND.

You may have to use a screwdriver or tweezers to make the motor cables curl up nicely to get the sensor board to sit flat against the top of the robot chassis. When the two front, and the two back mounting holes on the sensor board and the chassis line up, use the 10mm M3 screws to secure the board to the chassis.



IMG55 secure the PCB to the chassis.

Last thing to do is to put on the wheels. Take the o-ring and put it over the outside of the wheel so it is seated in the groove and form sort of a tire for the wheel. Now take a 5mm M2 and put it in to the small hole in the side of the wheel hub. Do not screw the screw all the way in, but only turn until it has started to enter the hole. Now you align the screw perpendicular to one of the flat sides of the motor shaft and push it onto the shaft. The screw side of the wheel should be on the inside of the wheel so that the flat side sits flush with the end of the motor shaft. Tighten the small screw until it pushes on the shaft. Once you have put on both of the wheels you can stand back and enjoy your handy work, because now the robot is complete!



IMG56 Fully assembled robot.