TILTIT - FINAL PROJECT REPORT

Haukur Hlíðberg, Kristófer Reykjalín Þorláksson &
Stefán Óli Valdimarsson

T-411-MECH
December 2015

School of Science and Engineering
Reykjavík University

Project report

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The first idea the group developed was to build a platform that could be used to play the maze game, then later developed into creating a platform that's manually controlled around six axes. For those who don't know the maze game, its objective is to get a ball through a maze and avoiding implemented obstacles by controlling the rotation around both the X and Y axes.

The team felt like the younger generation is getting more and more addicted to their new technology and we wanted to create something that the newer generation could play with, be inspired by and maybe assemble and develop themselves with appropriate manuals and education. The main obstacle the team faced was the software implementation because the mechanical part is pretty well understood and many open documentations available concerning calculations and how to successfully build a stewart platform.

The basic concept of a stewart platform is a *parallel manipulator*, which is just a fancy way of saying "a robot that moves a single platform". What makes stewart platforms distinct is the fact that they have six actuators that are used to control the movements of the platform around six axes.

Implementations of stewart platforms vary tremendously. The most prestigious ones are the Low Impact Docking System[1] (LIDS) that NASA is developing and flight simulators, like the one Icelandair[2] has, located in Hafnarfjörður. Both the aforementioned implementations are based on the basic concept of the stewart platform. Moving to another scale, the concept is also used in small projects similar to ours.

All the smaller projects that are available for purchase encourage you to do it yourself (DYI) which we will be doing as well. Some existing projects are the *Arduino controlled Rotary stewart Platform*[3] and *Full Motion Dynamics 6-DoF stewart platform*[4], to name a few. Those are either controlled exclusively by a serial console or fully automatic.

The idea was to make a project that could be seen as somewhere in between those two extremes by making a platform that is manually controlled with a controller. That way the platform isn't limited to a serial console and you can control it yourself, which is always more fun.

1.1 Background

Theoretical works on parallel mechanisms such as hexapods, date back many centuries, but where did parallel robots originate from? What can the RUMBA do and how does it function?

1.1.1 Stewart platform

Gough or Stewart platform? Many still fight about that topic even to day.

In the late 1950s a scientist called Eric Gough built the first hexapod that many refer to as the "stewart platform"[5], Gough developed his Tire-Testing Machine while he worked at Dunlop Rubber Co. A few years later, in 1965, D. Stewart published a paper where he describes a 6-DoF (multi-degree-of-freedom) motion platform that could be used as a motion simulator, *e.g.* a flight simulator. That same paper had a great impact on the field of parallel kinematics. Nothing further is known about D. Stewart. Then a few years later, in the mid 1960s, Klaus Cappel built the first flight simulator based on an octahedral hexapod after he had been requested to improve the existing 6-DoF vibration system by the Franklin Institute Research Laboratories in Philadelphia, which was, at the time, his employer.

These are the key members who helped shape the ideology of 6-DoF platforms. Even though a lot of people have had their hands in designing 6-DoF platforms, most of the time these platforms are referred to as stewart platforms.

1.1.2 RUMBA

The RUMBA (Reprap Universal Mega Board with Allegro driver) is an electronics solution for Reprap devices and some CNC devices. It operates with an ATmega2560 microchip so its primer functions are not that different from the Arduino Uno, which operates on an ATmega328P - almost plug and play, and therefore I2C and serial communications are ideal to send and receive data. It features 6 Pololu pin compatible stepper drivers and each one is replaceable if you destroy it. These drivers are pretty vulnerable to static electricity and other flickers in voltage and current. Each stepper driver has their own heat sink, because they tend to heat up pretty fast. It also features 7 ADC connectors for thermistors, 8 PWM (Pulse Width Modulation) connections, 4 I/O (input/output) pins, I2C and TX-RX pins. A word of advice: this device draws around 3000 mA constantly when all motors are in use, keep hands away and do not use thin wires for power. It's recommended to have a smoke detector with fully charged batteries nearby[6].

1.2 Requirements

After deciding what we wanted to do for this project, we sat down and outlined what we wanted the project to be and what we thought at the time was a realistic tasks to finish in this course. When we presented the first idea and goals to the instructor he told us they were unrealistic, discussed in detail in Chapter 2.1 The Design Phase. After that meeting the team reconsidered all ideas and the goals listed below were the outcome.

1. Smooth and accurate movements
2. Receive input from a controller
3. A user friendly controller
4. Transfer received input as G-code to the RUMBA
5. Move the platform according to the input from the controller

To simplify the problems solving process and increase efficiency, a FR-DP[7] list was made, see Table 1.1 .

	FR	DP
1	Controls manually	Controller
2	Process input from controller	Arduino module
3	Send signal between micro-controllers	Serial communications
4	Move accurately and fluidly	Transmission timings

Table 1.1: FR-DP list

FRs are the functional requirements and DPs are the design parameters. The relationship between FRs and DPs can be written as

$$FR = [A]DP \tag{1.1}$$

where [A] is defined as seen in equation (1.2).

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix} \tag{1.2}$$

The goal was for the A matrix to be an uncoupled design, but as any good designer knows, that is almost impossible, so we ended up with a decoupled design from equation (1.1), as seen in equation (1.3).

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \end{Bmatrix} = \begin{bmatrix} X & X & 0 & 0 \\ X & X & 0 & 0 \\ 0 & 0 & X & 0 \\ 0 & 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \end{Bmatrix} \tag{1.3}$$

Our analysis of the FR-DP matrix confirms what our instructor told us, the code and mechanical part will need our full attention.

Chapter 2

Methods

2.1 The design phase

Early on the instructor pointed out that this project would be complicated, both the mechanical and programming part. He suggested that we would build on a existing project. With that in mind, the search for an existing model began. There are all kind of designs for the Stewart platform out there, the one that was chosen is called *Stewart platform v2*[8]. It's a six armed manipulator, with six stepper motors controlled from a computer, see Figure 2.3.

The reason for the choice was the convenience of the project and the fact it was open source. The firmware and design was ready to use so we didn't have to spend the limited time we had making a complex code or mechanism. An assumption was made that the design and firmware worked perfectly, which later on we discovered didn't hold up. See Chapter 2.4.1 for more detailed information about the design.

When deciding on how to build on the already existing project the brainstorm technique was used. From research on the internet it was found that people have done many implementations on the Stewart platform, *e.g.* using a touchscreen[4] to automatically level the platform. That was seriously considered, however it was decided not to do that due to time limitations. Initially the only way to make the chosen design move was by sending a command through a serial console. Immediately we decided to change it so the platform could be controlled in other ways. After some discussion the execution we landed on was to make a controller that is able to control the movement of the platform using an Arduino.

When designing the controller we compared the two most obvious choices, the Playstation DualShock4[9] and the Xbox One Wireless Controller[10]. We leaned a little more towards the Xbox controller. After some sketches a final design was achieved that could be made using the laser cutter at Reykjavík University, see Figure 2.1.

The design has two analog sticks, one that would control movement in plane and the other the tilt, switches in the analog sticks and four switches on the front. The outcome can been seen on Figure 2.13. Because of the components the controller required a 12 wire connection to the Arduino. After consulting with Hrannar Traustason, supervisor of the electronics shop, it was decided that a VGA cable would be used to bridge the connection between the Arduino and controller. The assumption that the VGA cable contained 15 separate wires turned out to be wrong: we later discovered that didn't work out, see how in Chapter 2.3.5 Controller Test.

When designing the circuit inside the controller it was originally sketched on a paper and then later drawn in Altium, as can be seen on Figure 2.2. When picking out the right components the assumption was made that they all worked flawlessly.

The pseudo-code depicted in Algorithm 2.1.1 is a simplification of the loop function code. An example of how the coordinates are updated can be seen in Appendix A. The code in its entirety is accessible on GitHub: `https://github.com/reykjalin/TiltIt`.

Algorithm 2.1.1: LOOP()

comment: The general loop function always running on the controller

global All button states, debounce times *et al.*, see Chapter A
read all button states
if debounce time has passed
then {update relevant coordinates according to input from controller
if $buttonStateL2 = HIGH$
then {move motors to home position
if $buttonStateR2 = HIGH$
then {turn motors off
if $lastgstring$ **not** $gstring$
then {send new gstring to RUMBA

2.2 The build phase

After getting a hold of the schematics, we discovered they were drawn in a format that inventor, our software tool for the 3D schematics, did not compile with. Some effort was made, trying to export the files in to a format that the software could process, but were unsuccessful. The team felt like they were back in kindergarten when we discovered that the best and most sufficient way to get the schematics on the right format was to copy outer lines to the inventor sketch and trace it, one line at the time, through out all the schematics. The results of that can be seen in Figure 2.6.

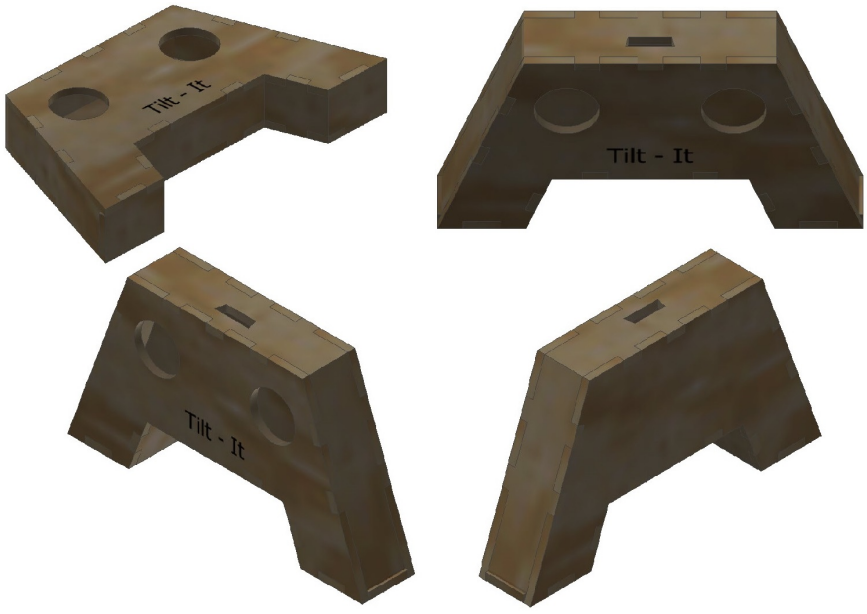

Figure 2.1: CAD drawing from various angles of the controller design

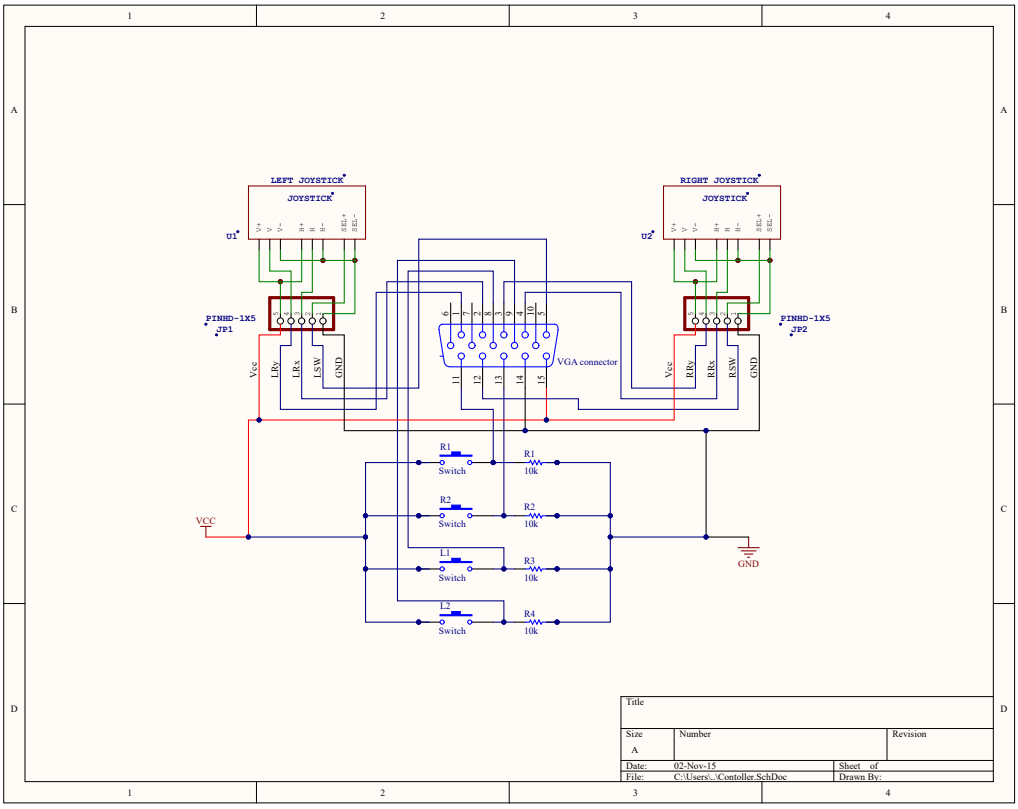

Figure 2.2: Altium schematic of the circuit inside the controller[11]

Figure 2.3: The original platform[8]

The next step was to decide the material. A couple of ideas were bounced around, such as wood and plexiglass, but after Reykjavík University bought a laser cutter, it was an obvious

choice, plexiglass it was and we would cut it on our own. That, however, turned out to be easier said than done.

The driver and job manager were relatively easy to set up, but when the actual cutting started the outcome was scaled parts that only scratched the material, so we sought the consultation of both our instructor Joseph T. Foley and Indriði Sævar Ríkharðsson. It turned out that in order to cut the material, the line width had to be smaller than 0.025 mm, but never the less, it still scaled almost all our drawing by a random factor, except very small ones. After countless hours, creative brainstorming and endless headaches, it turned out to be the PDF viewer, Adobe reader, whom was to blame. The viewer scaled everything to fit the prefixed paper size and after setting the size to ISO standard that fitted the cutter everything went smoothly.

It turned out that in order to cut the material, the line width had to be smaller than 0.025 mm, but nevertheless, it still scaled almost all of our drawings by a random factor, except very small drawings.

After countless hours, creative brainstorming and endless headaches, it turned out to be the PDF viewer, Adobe reader, who was to blame. The viewer scaled everything to fit the selected paper size and after setting the size to the ISO standard that fitted the cutter everything went smoothly.

At this moment all components were in our possession except the RUMBA motor controller, so the assembly began. Half way in, when the base was ready as shown in Figure 2.5, we had to do a bit of research on how to attach the ball joint to the arms. It turned out that Tómstundahúsið[12] is selling ball joints with exactly the right bolts to fit the arms and bearing mounts on the platform it self.

2.3 The testing phase

This project consisted of many individual parts that could be tested separately, and yet most of them had one thing in common: The RUMBA[6].

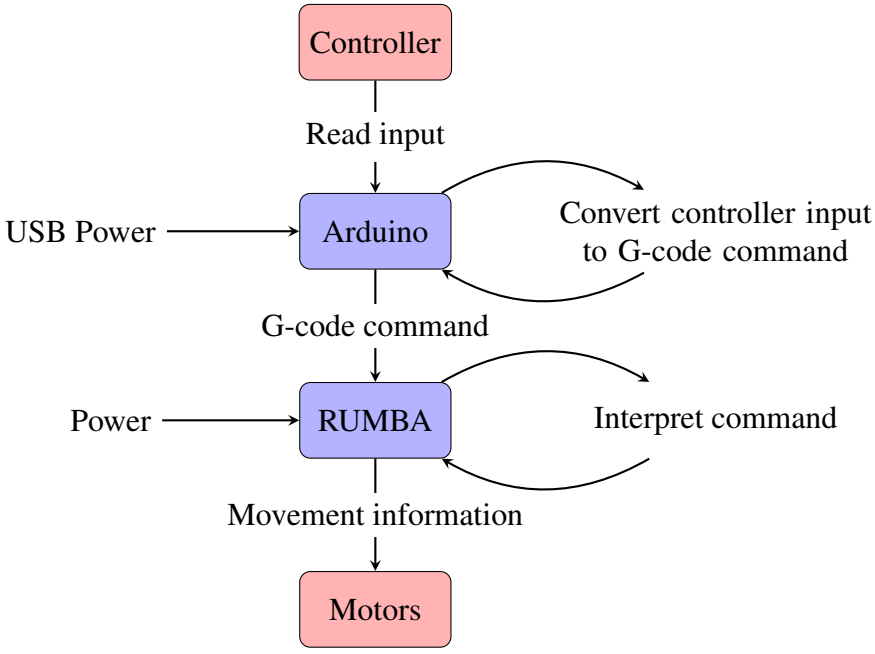

Figure 2.4: System digram

Once we had the RUMBA we immediately began testing the motors, see Chapter 2.3.1 Motor Tests. As soon as we were sure they worked, we tried controlling them with an analog stick. That involved reading the signal from the stick on an Arduino and sending that signal to the RUMBA, thus creating the need for a communication test, see Chapter 2.3.2 Communication Test.

After we had the basics of the RUMBA down we assembled the rest of the platform and tried hooking everything up. While testing the firmware, see Chapter 2.3.3 Firmware Test, we ran into a bit of trouble with the pairing of the motors with the endstops switches so we did the endstop button tests, see Chapter 2.3.4 Endstop Button Tests.

After we got those things working we moved on to testing the controller, as described in Chapter 2.3.5 Controller Test, were we ran into some significant troubles.

2.3.1 Motor Tests

The first thing we did after getting the RUMBA was to connect a motor and test, see it move and how it worked.

We programmed the RUMBA so it would spin the motor 360° clockwise, then 360° counter-clockwise, however once we started the test we noticed that it was actually going two full circles, or 720°, each direction. After a while we realized the firmware[13] running on the RUMBA was optimized for stepper motors that had 400 steps per turn, while ours have 200 steps per turn. We optimized the RUMBA for 200 steps per turn and after that the motors spun 360° clockwise, then 360° counter-clockwise, like they should.

After testing one motor, the next thing we did was connect all 6 motors driving the platform to the RUMBA and ran the same test on all 6 motors, so the RUMBA would turn a motor one full circle clockwise and another counter-clockwise and then move on to the next motor.

After getting them to work we connected an analog stick to the motors and controlled the movement of an individual motor using the analog stick to turn each motor.

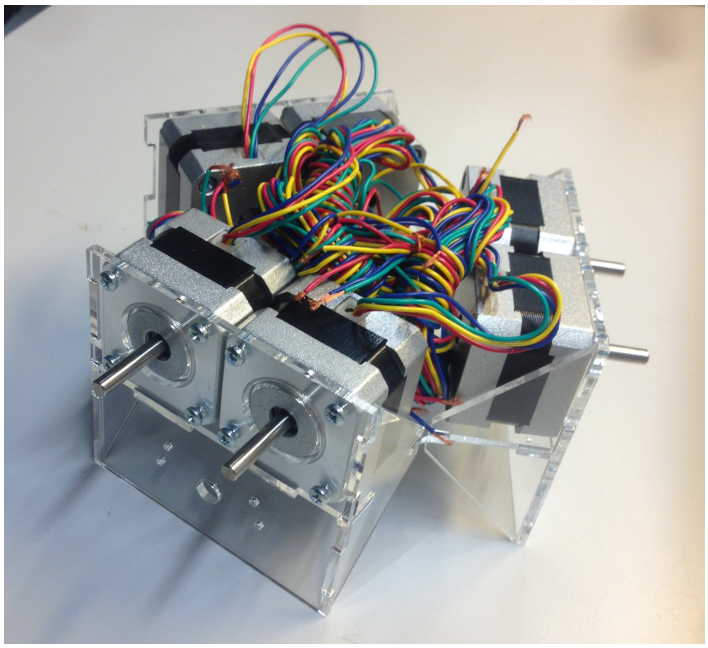

Figure 2.5: The base of the platform assembled

2.3.2 Communication Test

When testing the motors the last test with the analog stick involved reading the analog stick on an Arduino and sending the signal to the RUMBA through I2C. Everything worked great and it seemed very responsive. We tried a couple of different speeds, see Table 2.1, and the movements were very smooth when moving one step every 100ms.

Even though this worked great, eventually, during the controller tests, we decided to send the coordinates through serial communications instead of I2C due to the way the RUMBA receives G-code commands as strings.

2.3.3 Firmware Test

Testing the firmware wasn't to bad. Everything seemed to work great out of the box after setting it to work with stepper motors that have 200 steps per turn. It was while testing the firmware that we found out the endstops weren't correctly wired according to the motors. For more details on that see Chapter 2.3.4 Endstop Button Tests.

After fixing the endstops we began testing every aspect we could think of regarding the firmware. It was at this stage of the testing were we encountered a major issue: the platform wouldn't tilt correctly. When going from the zero position, or going through it, the platform tilted correctly, but otherwise it had to shoot back up to the zero position before tilting to the new coordinates.

We spent quite a bit of time trying to figure out what was causing the issue, but eventually we contacted the developer of the firmware and found out it was a bug in the firmware. He fixed for us within a measly two hours after submitting the issue and everything worked great after that. Many thanks to Dan Royer, the developer of the firmware.

2.3.4 Endstop Button Tests

After putting the platform together and while testing the RUMBA firmware we found that the homing function didn't work. The motors would sometimes look like they were jammed and stopped moving before they hit the endstops. We figured the switches might be broken or improperly paired with the motors.

The switches were quite easily tested, the only thing to test was if the signal was getting trough to the RUMBA. After making sure each switch worked we looked at the connections to the motors. After seeing which switch stopped which motor we wired them up correctly and everything worked.

Sending speed [ms]	Jerkiness
500	10
400	8
300	5
200	3
150	1.5
100	1.2
50	4

Table 2.1: I2C sending speed in milliseconds and jerkiness on a scale of 1-10 (roughly estimated) where 1 is smooth (no jitter) and 10 is very jittery

2.3.5 Controller Test

To say that the first controller test was a disaster is probably an understatement. After connecting it via VGA cable to the Arduino every thing worked according to plan except 3 buttons, L1, R1 and R2 gave the same signal.

Everything was disconnected and measured but every single test had the same result, there were no short connections in the circuitry. The next idea that popped up was whether the Arduino was still fully operational, but after testing the same signal on 3 different Arduinos, the Arduinos were ruled out as a source of this chaos.

After banging our heads against the wall for several hours, a discovery was made: the VGA cable had 5 of its pins connected to the same node, shorting these 3 buttons.

The next test, now tested with a custom made cable with a VGA plug on one end and pinheads to plug directly to the breadboard next to the Arduino, showed promising results until it was discovered that L3 and R3 were giving gibberish signals. After connecting each one to an oscilloscope, the conclusion was that we had corrupted them by overheating them while they were being soldered. In the end we didn't fix the L3 and R3 buttons since the yaw motion (twisting around a vertical axis) isn't necessary, and programmed the L2 and R2 buttons as the initialize button and kill switch respectively, see Table 2.4.

The controller was connected to the Arduino and it connected to the RUMBA, as described in Chapter 2.4.2 Instructions. The RUMBA receives G-code[14] commands through serial communications from the Arduino. At first we tried sending the new coordinates from the controller through I2C but an issue with the firmware made that very hard. Eventually we decided to simply use serial communications and that seemed to work great. The only issue seems to be related to sending speed and the jerkiness of the platform due to that speed. To fix that we probably have to send as little information as we can through I2C and have the RUMBA interpret that in an attempt to make the process more seamless.

2.4 Usage

Since this particular implementation of a stewart platform doesn't have many applications for the average person a device like this is mostly aimed at people that want to try assembling and configuring a robot, improve it, or design something entirely different based on this design.

Even if you have no prior experience working with electronics or robotics assembling and using the platform shouldn't be impossible, on the contrary it could even be simple. If you follow the instructions in Chapter 2.4.1 you should be able to properly assemble the platform. Once it's assembled just connect it to power and have some fun!

If you're interested in doing something with the platform a good place to start is to learn some programming in C++ and mess with the configuration of the controller. You can get the code from GitHub, see Appendix A for a link. Once you're comfortable with that you can learn more about programming and change the firmware on the motor controller, or you can read and learn about electronics and improve or add to the circuitry.

Once the platform is assembled and working no routine tests or calibrations are needed. Just make sure to handle it with care to prevent any damage that might occur when the platform is mishandled.

Just keep in mind that all you need is curiosity, and a little common sense, and assembling and using a project like this can be a very fun and informative, experience.

2.4.1 Installation

The parts included in the kit can been seen in Table 2.2. The kit would be packaged in wooden box with eco-friendly standards. The only thing not included in the kit that the buyer needs is a tube of strong glue, preferably transparent.

The first thing to do after unpacking is to glue together all the parts according to the manual. A complete set of cut out parts are shown in Figure 2.6. After the glue has dried it's time to place the stepper motors, switches, arms, ball joints and rods. The stepper motors and switches are tightened to the base of the platform, as shown in Figure 2.5, and the arms are placed on the motor shaft. Make sure to use the spacer when placing the arms to ensure symmetry of the platform. The ball joint housings are fixed on each end of the rods and mounted on the ball joint pins. The pins are then glued to the top fastening and the arms on the motor. At this time the base should look like Figure 2.7. Last but not least is gluing the top of the platform to the top fastenings. Now the assembly is done and should look like Figure 2.8.

Now that the platform itself is fully assembled it's time to connect the motors and switches to the RUMBA motion controller. The most convenient way is to pick a motor and call it "motor 1". Then going a counter clockwise circle and continuing numbering all the six motors, as shown in Figure 2.11. Each switch get the same number as the motor placed above it. After numbering you can connect the motors and switches to the RUMBA as shown in Figure 2.12. When connecting the motors be sure to connect in the following color order: red - green - blue - yellow. The switches should be connected in a way the red cord faces the outer rim of the board and blue cord faces inwards.

When the platform is connected to the RUMBA the only thing left is to connect the remote controller to the Arduino and the Arduino to the RUMBA. Connecting the Arduino module to the RUMBA board is simple. Connect the Tx pin on the Arduino to the Rx pin on the RUMBA and make sure they are grounded together. When connecting the VGA cord to the Arduino the pins as shown on Figure 2.9 are plugged in to the breadboard, make sure that the pins aren't connected to the same node. Use the color code as listed in Table 2.3 to connect the correct pins on the breadboard to corresponding Arduino pins as shown in the aforementioned table.

There are two option recommended when powering the Arduino. First there is USB power which requires an external device to provide the USB port, the seller does not provide

Part	Quantity
A plate with all the parts cut out	1
Custom made controller	1
Stepper motors from Sparkfun[15]	6
RUMBA motor controller[6]	1
Power supply 12V, 3A	1
Arduino Uno	1
Switches[16]	6
Ball joints	12
Rods	6
Screws	42
Nuts	24
Manual	1

Table 2.2: Parts included in the kit

such a device. Secondly using a 9V battery, in that case the cathode (+) is connected to the V_{in} pin and the anode (-) is connected to the GND pin. A battery is not included in the kit. At this point all the modules should be connected together and ready to go. If for some reason the platform is not working properly some instructions can been found in Chapter 2.4.2 Instructions.

2.4.2 Instructions

After all the modules are connected and the platform is fully assembled, initialize the platform by simply pressing L2. No calibration is needed, the correct software is already uploaded to the boards. In Table 2.4 there is a summary of what each button on the remote does, in Figure 2.13 you can see the placement of the buttons on the controller. To see an example video of how the platform is operated follow this link `https://www.youtube.com/watch?v=LdpKwXdAqzQ&feature=youtu.be`.

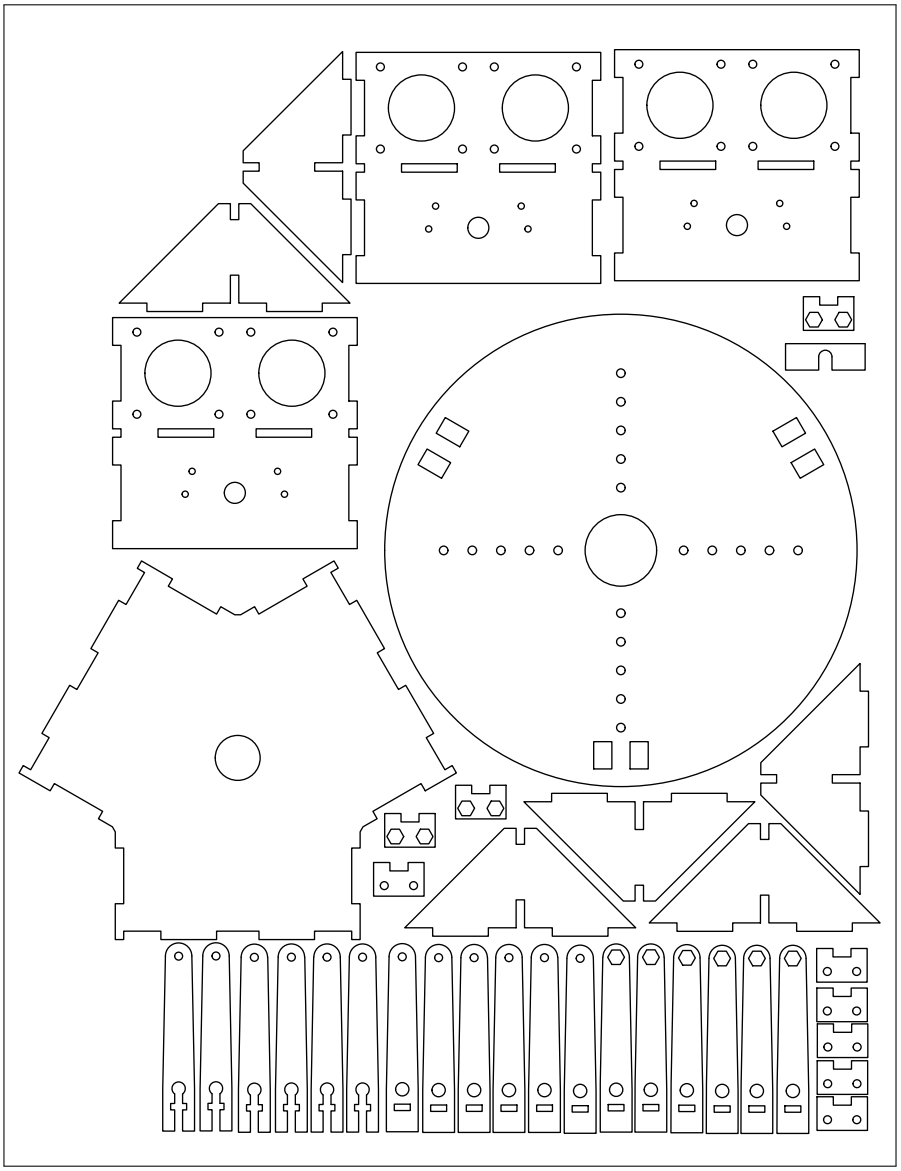

Figure 2.6: The cut out plate included in the kit

The only maintenance necessary is to keep an eye on the arms mounted on the shaft of the motors, they can get loose after some time because of the vibrations of the platform. This can be checked by powering up the platform and lightly wiggling the arms, if the arms move they should be tightened.

When operating or assembling the platform there are a some things to keep in mind. First of all when working with the Arduino or RUMBA boards be sure to be properly grounded

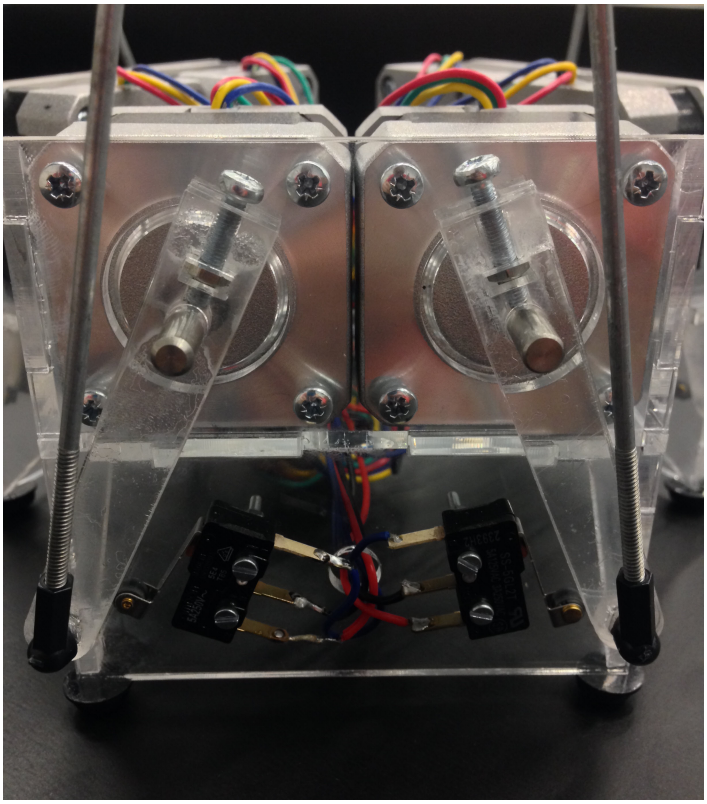

Figure 2.7: The lower part of the platform assembled

VGA Pin Number	Color	Button	Arduino pin
1	Black	LY	A0
2	Blue	LX	A1
3	Turquoise	RY	A3
4	Brown	RX	A2
5	Green	L3	A4
6	EMPTY	EMPTY	EMPTY
7	EMPTY	EMPTY	EMPTY
8	White	L1	3
9	Red	L2	2
10	EMPTY	EMPTY	EMPTY
11	Yellow	R1	6
12	Grey	R3	A5
13	Pink	R2	4
14	Orange	GND	GND
15	Purple	Vcc	5V

Table 2.3: VGA connections and colors. This table relates to Figures 2.9, 2.10 and 2.13

to avoid static shocks that could damage the hardware. This can be avoided by using a anti-static wristband[18].

A good addition to the platform is a cooling fan. After a long operating time the stepper motors could get hot and possibly melt the glue that holds the platform together, however, by using a fan this can be avoided.

Right analog stick	Movement in xy plane
Left analog stick	Change in roll and pitch
R1	Increase z-axis
R2	Kill switch
L1	Decrease z-axis
L2	Initializes the platform

Table 2.4: Corresponding actions to each button

If something goes horribly wrong first thing to do is to cut the power. Generally the first component to be damaged, if any, is one of the stepper controllers on the RUMBA. But luckily those are easily replaceable, they are pretty much plug and play. The damaged one is plugged out of the board and the replacement is plugged in, easy as pie. If you encounter any problems with the platform please contact Kristófer R. Þorláksson via email: kristoferrt@gmail.com.

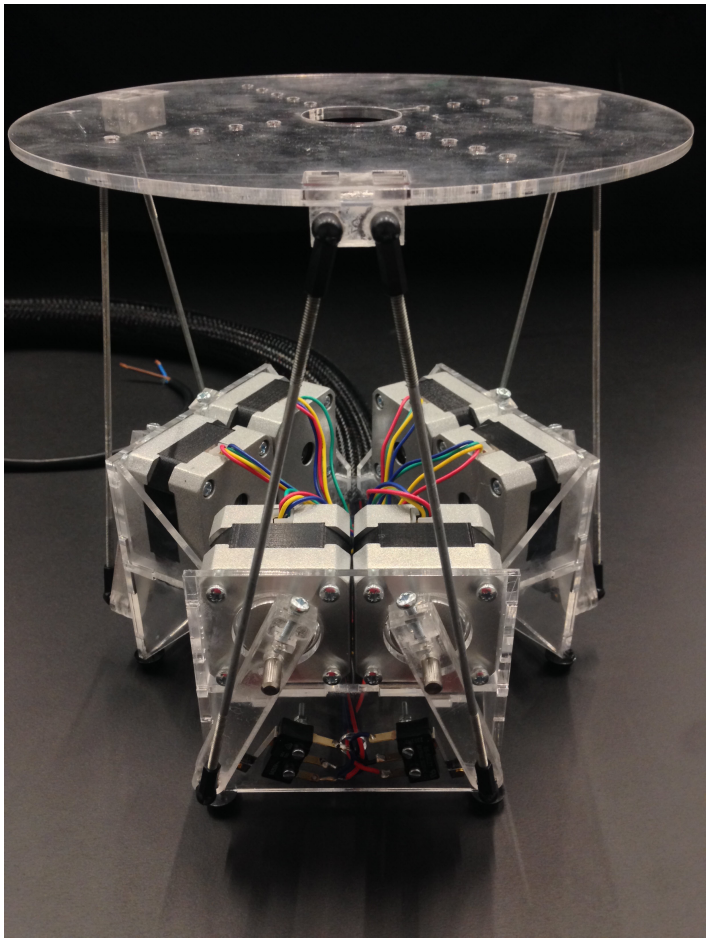

Figure 2.8: One side of the complete assembly

2.4.3 Safety

When working with electronics make sure that you proceed with great caution. The system is running at 12V and 3A and that produces a lot of energy, 36W to be exact, so when the system is on never touch exposed wires, doing so can be very dangerous, even lethal in extreme cases.

Before you change or reconnect anything in the circuit turn the system off, or you risk accidentally short-circuiting delicate components like the components on the RUMBA, the RUMBA itself or the Arduino.

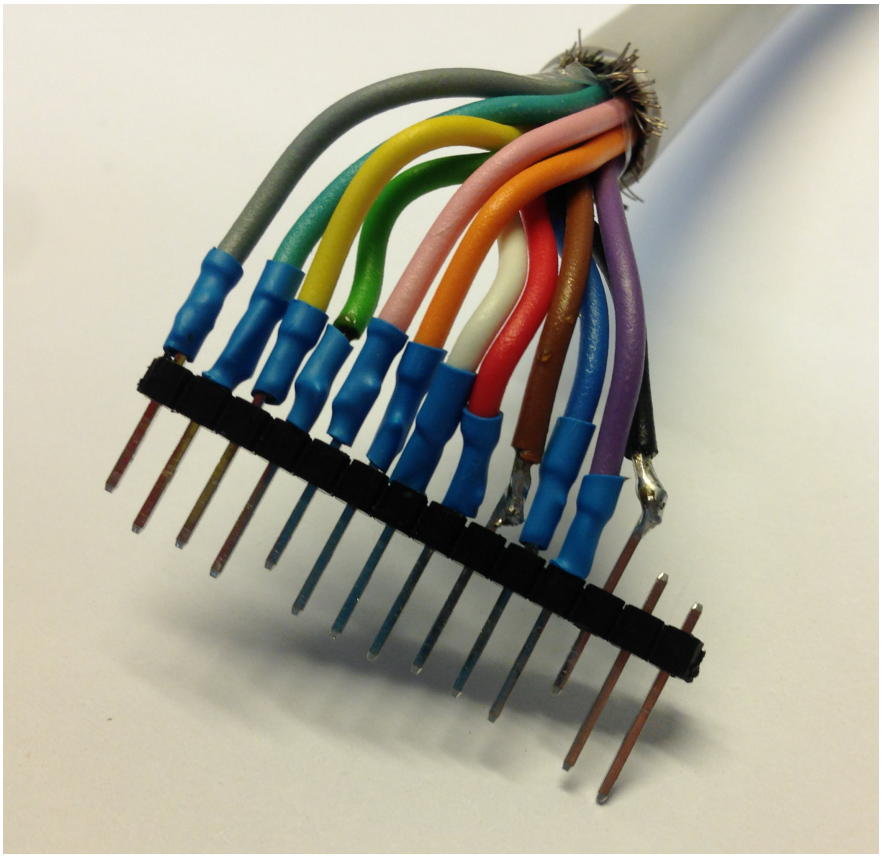

Figure 2.9: Pins on the other end of the VGA cable. Refer to Table 2.3 for color codes.

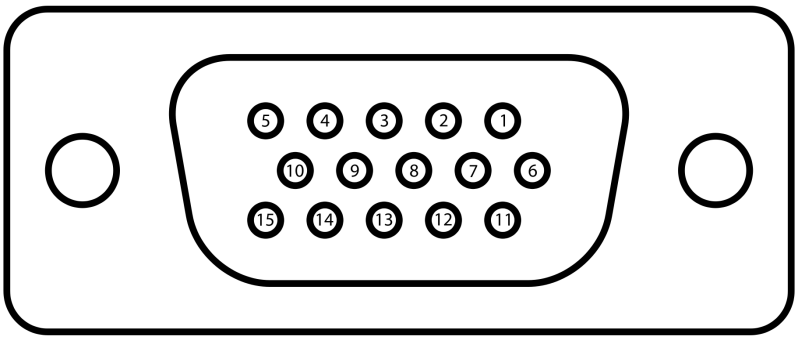

Figure 2.10: Numbering on the VGA connector[17]. Refer to Table 2.3 for pin connections to the cable.

Make sure you watch the temperatures of the motors, they tend to get hot, in extreme cases even hot enough to burn skin if you're not careful.

When working with the circuit components watch out for static electricity. If you touch some of the more delicate components and your body has some static electricity built up that can fry the components, and you'll have to buy new ones.

Make sure you don't hit the USB sockets with any wires, that can cause a short-circuit in your computers USB port, which can destroy the port itself.

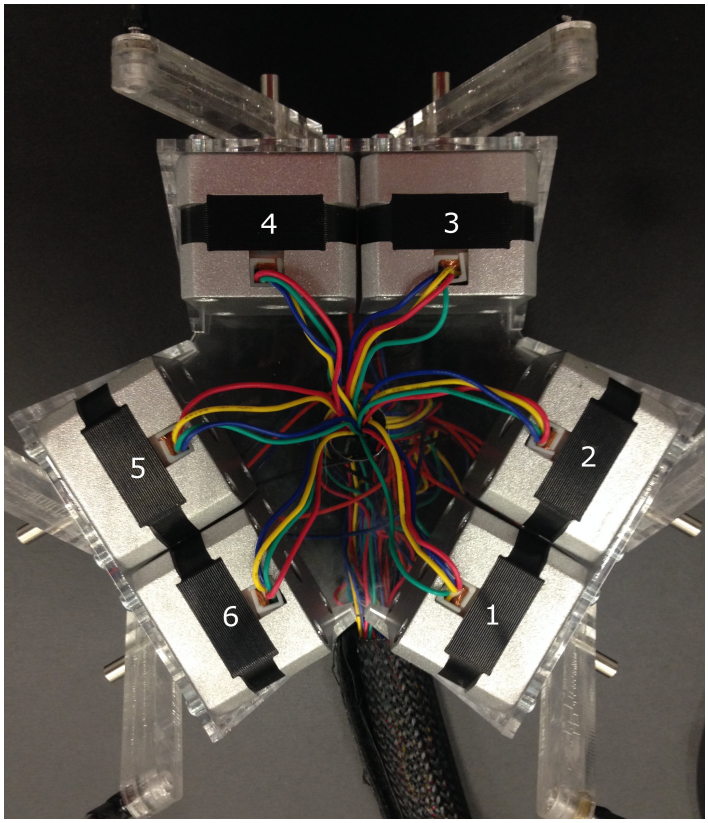

Figure 2.11: Top view of the motors

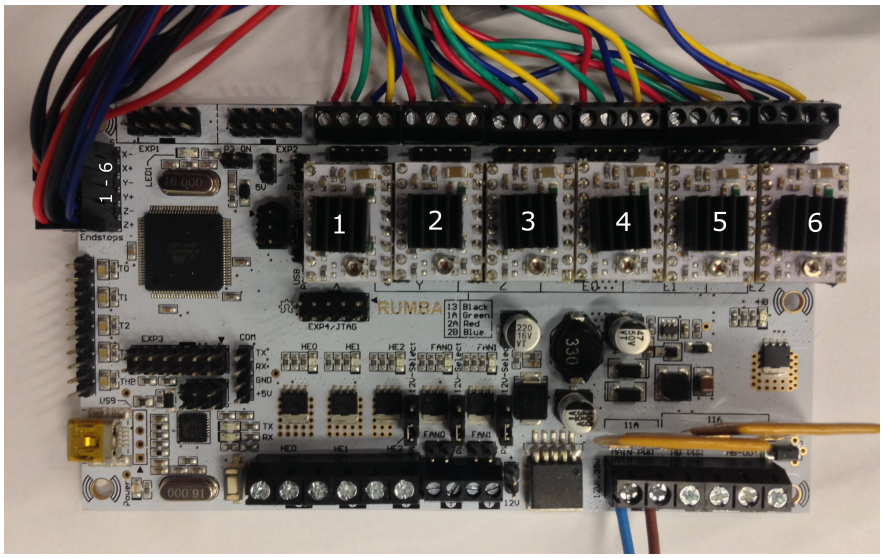

Figure 2.12: Numbering for the RUMBA

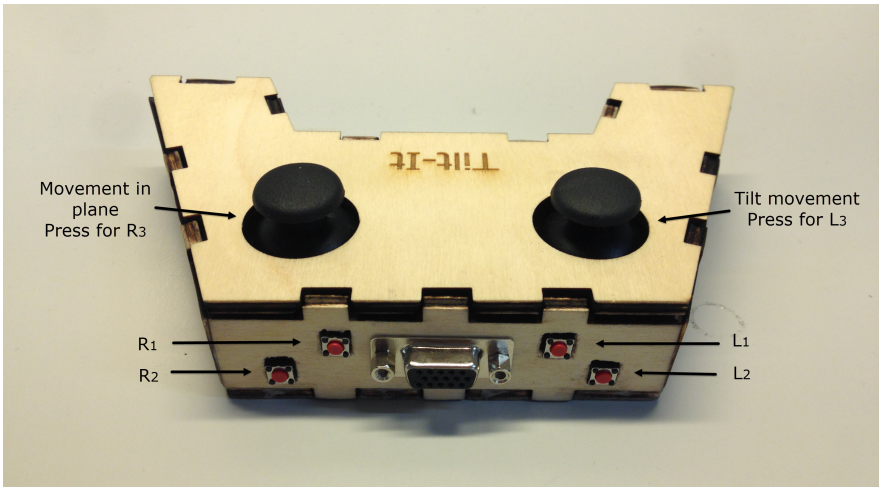

Figure 2.13: Every button on the controller with unique labels

Chapter 3

Results

3.1 Hardware

In this section we will be discussing all the hardware errors the team encountered while designing the platform and the process to getting a fully functional item.

3.1.1 The laser cutter

As discussed earlier on in Chapter 2.2, we ran into a rather frustrating problem when the base was ready for production. The laser cutter didn't cut and scaled almost every object we tried to cut out. After reading few chapters in the manual and consulting with Indriði Sævar Ríkharðsson, lector at RU, we finally figured out it was a schematic issues, not software.

First of all it didn't cut because of the line width had to be smaller than 0.025 mm, and a standard layer in inventor is 0.51 mm. The scaling problem turned out to be a rather obvious, but easily overlooked. We always converted the schematics in to a PDF file, and the PDF viewer scaled all of the schematics that didn't fit into the preset sheet size. By setting a larger sheet size and unchecking the auto fit to sheet box all our problems were ancient history and the fun could begin.

3.1.2 Motors and switches

When telling the platform to go to the home position it would sometimes get stuck and we'd have to push the arms that got stuck manually to the endstop switches. It didn't take a lot to figure out this was due to the switches being improperly paired with the motors. That meant that when an arm hit an endstop switch it wasn't necessarily the endstop for that particular arm, so it would in fact stop another arm somewhere else on the platform.

To fix this we simply mapped the places of the switches relative to the motors and wired them together following that order. It took a bit of time since each switch has three terminals and we had to figure out which motor each switch was affecting, but after writing everything down it was pretty clear which switch went to which motor.

3.1.3 The VGA cable

The VGA cable played the team for a fool. When we were testing and mapping the controller, we discovered that R1, R2 & L1 gave the same signal in the test script written to test the controller functionalities. The cable itself was the last place we thought to look at. It turned out that five of the pins were connected to the same pin within the VGA cable. Not

even the TA suspected the cable to be the source to the problem the team faced, a human error was considered to be much more likely. Note to self: Always do your homework on all manufactured components in your project before connecting everything and hope for the best, and as a vice man once said "RTFM".

The day after we got a cable sponsored from Smith&Norland, a 12 wire cable, that we mounted to a VGA plug from the storage room in V207 and at the other end we soldered pinheaders that we plugged straight to the breadboard on the Arduino.

3.2 Software

While testing the platform we encountered quite a number of problems as mentioned in Chapter 2.3 The Testing Phase. The first real software related problem we encountered was when we realized the platform had an issue with tilting correctly. It always wanted to shoot back to the home position before adjusting the angle. This turned out to be a glitch in the firmware running on the RUMBA, as mentioned in Chapter 2.3.3 Firmware Test.

The other real software issue we encountered was when we realized the serial communications were too slow to handle the amount of information we were sending through the lines. This resulted in jerkiness when moving the platform. At this point time was not on our side, we had a time consuming idea how to solve it whit I2C. In order to do so we would have to send as little data as possible and construct the G-code on the motor controller module as briefly mentioned in Chapter 2.3.5

3.2.1 Tilting errors

When we had the controller working and started trying it out we noticed that the platform, when tilted, always wanted to go to the home position before adjusting the angle. We spent a couple of hours, split amongst a few days trying our best to figure out what was causing this problem, and after trying, and failing miserably, our instructor mentioned that we should probably have contacted the developer of the RUMBA firmware[13] as soon as we noticed this problem. So we asked for help on the developers GitHub page and within 2 hours the problem was fixed! Again: many thanks to Dan Royer, the developer of the firmware.

3.2.2 Serial communications too slow

Initially we wanted to have the Arduino connected to the controller communicate with the RUMBA via I2C, but we ran into some difficulties getting it to work. The information was passed through correctly but since the firmware is built to accept strings and I2C can only send unsigned integers, converting the data to a form the firmware understood proved to be quite difficult. Eventually we decided to use serial communications since it's very simple compared to the I2C code and you can send strings.

When we had the controller and platform fully functional and we began optimization of the transmission intervals we noticed that the serial communications couldn't send information fast enough for the platform to feel responsive and move smoothly at the same time. This means that while the platform moves fast and feels responsive the movements aren't close to being as smooth as we want them to be and the platform jerks around when moving, as can be seen in a video of the platform operating here: `https://youtu.be/LdpKwXdAqzQ`.

Unfortunately we didn't have the time to properly address this issue, but we did optimize the platform to be as smooth as possible. Fixing this is part of future work surrounding this platform.

Chapter 4

Discussion

4.1 Conclusion

1. Controls manually:

   The controller seems to be working fine after resolving the cable issue, except for the problem with the left and right analog stick switches mentioned in Chapter 2.3.5. If we manage to fix the sending speeds that would increase the accuracy and sensitivity of the controller at the same time. We'd probably have to get new analog sticks to fix those buttons though.

2. Input processing for controller:

   When moving the platform around it has certain limitations. When you go outside those limitations it circumvents them by "overflowing" the coordinates, causing the platform to move in very strange ways. As we found out during stress tests when messing with the sending speeds, if you send enough information before/during the overflow it will actually move in very strange ways for quite a while until settling down.

   To fix this we'd have to improve the restrictions we've set on the movement. That, however, is a very complex task since the restrictions aren't uniform around the whole platform. Improving these limitations would include making each coordinate dependent on all the other coordinates. Lets say you want to move the platform up (along the z-axis), what you'd have to do is the following:

   a) Check x-coordinate, if it's outside restrictions, decrement.
      i. Are all other coordinates OK? If not fix.
   b) Check y-coordinate, if it's outside restrictions, decrement.
      i. Are all other coordinates OK? If not fix.
   c) Check roll, if it's outside restrictions, decrement.
      i. Are all other coordinates OK? If not fix.
   d) *et al.*

   This is definitely not going to speed anything up and will only cause the code to be overly complicated.

   This could possibly be solved if you don't restrict the platform that much, but then again, you're probably going to encounter edge cases where the platform will go crazy

anyway. Thus, for the sake of speed and good response times, we decided to not restrict it more than we have already.

3. Communication between micro-controllers:

   We think the serial communications lines aren't able to properly keep up with the speed of the transmissions and that seems to be the root of the problem.

4. Smooth and accurate movements:

   Currently the system is functional, plane movement is fairly responsive and accurate as well as tilting movement and complex movements are functional until the platform reaches the undefined region, but we'd like to have the movements smoother. If we could send the information faster we'd probably be able to smooth out the movements. Unfortunately we didn't have the time to do that, but that's definitely some future work to be done!

The motors get very hot when operated for more than roughly 15-20 minutes, so we'd like to put in some cooling system. Putting a fan above the motors keeps them very cool so creating a mount for the fan is one option. Maybe not the most elegant solution but it's definitely practical and cheap!

4.1.1 Accomplishments

As previously mentioned, a stewart platform is rather hard project for undergraduate student to accomplish in 12 weeks. With that in mind we decided to define the lowest level of success to complete mechanical assembly, wiring everything together and be able to do the basic movements.

But we wanted more, so we designed a controller and successfully connected the two together. To be fair, the rest of the requirements are all about tuning the controller and you can only tune things so much, and only according to your preferences.

Some of the key requirements that we feel have been successfully fulfilled are listed in Table 4.1. All in all the project was a success, even though there are plenty of things we'd want to improve.

Tasks finished	Unresolved issues
Building a platform	Faster communication
Wiring all peripherals	Smoother movements
Connecting modules together	Wireless controller
Constructing G-code from controller	Additional gameplay feature
Tune response time on signal processing	

Table 4.1: List of finished and unfinished tasks

4.2 Future improvements

One of the big hassles when operating the platform is the cable going from the controller to an Arduino. It would be amazing if we could make it wireless! It shouldn't even be that hard as long as we're willing to modify the controller a bit and design a PCB to fit within it.

To make the controller wireless we'd have to move an Arduino (or a specialized microcontroller) into the controller itself so we don't need any wires outside the controller. Getting rid of the wires isn't all though, a wireless module is needed to communicate between the Arduino and the RUMBA. A wixel[19] or a bluethooth module would probably do the trick. That, however, brings back another issue: how fast can you communicate using these modules? If they aren't faster than the serial communications lines or - what could be worse - *slower* than the serial communications, that would probably mean we'd have to have the controller wired.

In the end, we haven't tried to make it wireless so we don't actually know how it'd work. Trying to make it wireless would, nevertheless, be an interesting project to take on.

When thinking about future possibilities the end result will always be same: your only real limitation is your ability to come up with scenarios where a platform like this could be useful. To overstate this a little bit: your options are pretty close to endless.

Improving the current design and making the platform bigger is another thing that could be extremely interesting. Specializing the design to simulate some sort of movement, *e.g.* earthquakes, driving, ships out on sea, flight, *et al.* could be very rewarding, or you could go the exact opposite direction and have the platform work against movement. Optimizing the platform to keep something stable while in flight, out on see or driving could be a very helpful thing to have in many situations, *e.g.* helipads on ships or even medical operations out on sea, in flight or while driving. In short: whether you're going out to sea, a family holiday in your car or simply at home watching your kids, a stewart platform will, probably, come in handy!

Improving the design would take a lot of time, mostly because a robot moving a platform around six axes is a very complicated thing. There are entire theses based around the most efficient ways to move a platform like this, *e.g.* the paper D. Stewart wrote in the 1950s on 6-DoF motion platform[5] and many more based on it. Not only would it the design take a lot of time, but you'd also have to build the thing. Creating a big project based on this, *e.g.* a helipad on a boat, would not only be time consuming, it'd also be extremely expensive. With a 3-man team a project like that might take around 400 hours per person, give or take a couple hundred hours depending on the size of the project.

Proper improvements on the code running on the controller could probably be done in less than a week, around 10 hours of work, but improving the firmware running on the RUMBA[13] could take weeks, our best estimation is around 100 hours, without the help of the people that wrote it.

Bibliography

[1] P. Dunlap, S. Bruce, and C. Daniels, *Overview of lids docking seals development*, Developement presentation, Nov. 2008. [Online]. Available: http://ntrs.nasa.gov/search.jsp?R=20130013108.

[2] I. Ingólfsson, *Nýr flughermir icelandair tekin í notkun*, Nov. 2015. [Online]. Available: https://www.youtube.com/watch?v=JtpUE8i-8EM.

[3] ThomasKNR, *Arduino controlled rotary stewart platform*, 2015 November. [Online]. Available: http://www.instructables.com/id/Arduino-controlled-Rotary-Stewart-Platform/.

[4] Full Motion Dynamics, *Home of the 6 degree-of-freedom motion simulator, a senior project in mechanical engineering at SJSU*, Nov. 2015. [Online]. Available: http://www.fullmotiondynamics.com/.

[5] Parallemic.org, *The true origins of parallel robots*, Nov. 2015. [Online]. Available: http://www.parallemic.org/Reviews/Review007.html.

[6] RepRapWiki, *RUMBA*, Nov. 2015. [Online]. Available: http://reprap.org/wiki/RUMBA.

[7] N. P. Suh, *Introduction to axiomatic design*, Presentation from MIT, Oct. 2015. [Online]. Available: http://ocw.mit.edu/courses/mechanical-engineering/2-800-tribology-fall-2004/lecture-notes/ch10_axiomatic.pdf.

[8] MarginallyClever, *Rotary Stewart Platform v2*, Author: Dan Royer, Sep. 2013. [Online]. Available: https://www.marginallyclever.com/shop/stewart-platforms/rotary-stewart-platform-v2.

[9] Sony Computer Entertainment America LLC, *Dualshock®4 wireless controller*, Nov. 2015. [Online]. Available: https://www.playstation.com/en-us/explore/accessories/dualshock-4-wireless-controller/.

[10] Microsoft, *Xbox One wireless controller*, Nov. 2015. [Online]. Available: http://www.xbox.com/en-US/xbox-one/accessories/controllers/elite-wireless-controller.

[11] Adafruit, *2-axis-joystick-breakout-board-with-mounting-holes*, Sep. 2013. [Online]. Available: https://github.com/adafruit/2-axis-joystick-breakout-board-with-mounting-holes.

[12] Tómstundahúsið, *Netverslun tómstundahússins*, Nov. 2015. [Online]. Available: http://www.tomstundahusid.is/.

[13] D. Royer, *RotaryStewartPlatformV2*, Nov. 2015. [Online]. Available: https://github.com/MarginallyClever/RotaryStewartPlatform2.

[14] RepRapWiki, *G-code - RepRapWiki*, Nov. 2015. [Online]. Available: http://reprap.org/wiki/G-code.

[15] SparkFun.com, *Stepper motor with cable*, Oct. 2015. [Online]. Available: www.sparkfun.com/products/9238.

[16] MarginallyClever, *Switch, micro SPDT, 15A, V-152-1C25*, Nov. 2015. [Online]. Available: https://www.marginallyclever.com/product/switch-micro-spdt-15a-v-152-1c25/.

[17] Mobius, *VGA connector - Wikipedia*, Picture made by the user editing the Wikipedia article, Jun. 2006. [Online]. Available: https://commons.wikimedia.org/wiki/File:DE15_Connector_Pinout.svg.

[18] mctsolchris, *Anti-static wrist straps*, Nov. 2015. [Online]. Available: http://microcentertech.com/byopc/?p=73.

[19] Pololu, *Wixel Programmable USB Wireless Module*, Nov. 2015. [Online]. Available: https://www.pololu.com/product/1336.

Appendix A

Code

Examples of the code running on the controller for communications with the RUMBA. The code is available in its entirety on GitHub: `https://github.com/reykjalin/TiltIt`

Listing A.1: Controller module: example from loop function

```
/**** Change 'X' every debounce interval ****/
if((micros() - lastDebounceTimerx) > incrDebounceDelay){
    if(buttonStaterx < 450){
        if(X < 5){
            X+=0.015;
            gstring += " X" + String(X);
        }
    }
    else if(buttonStaterx > 550){
        if(X > -5){
            X-=0.015;
            gstring += " X" + String(X);
        }
    }
    lastDebounceTimerx = micros();
}
/**** Change 'X' every debounce interval ****/
```

School of Science and Engineering
Reykjavík University
Menntavegur 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.ru.is