People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

M'Hamed BOUGARA University – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment for the Requirements of the Degree of

MASTER

In Electrical and Electronic Engineering
Option: Computer Engineering

Title:

Design and implementation of an autonomous hexapod robot with obstacle avoidance and object finding/tracking

Presented by:

- TELLACHE Mohamed Mounir

Supervisor:

- Mr. NAMANE Rachid

Registration Number:...../2016

To my parents, my grandfather, my grandmother, family, friends and of course you.

Acknowledgment

In the name of Allah, the Most Gracious and the Most Merciful. Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis.

I would like to express my eternal gratitude to:

My parents, without their constant support and guidance none of this would have been achieved.

My supervisor Mr.NAMANE, for all the time and effort he spent guiding me not only during the project but through the years.

Mr. KHALDOUN for his help & guidance

Mr. METEDJI for his encouragement and the power supply.

All teachers and instructors at the Institute who helped me during those five years.

The carpenters who helped me refining the first robot.

No acknowledgement would be complete without expressing our appreciation and thankfulness for the maintenance laboratory workers for their support and help.

Abstract

The aim of this project is the design and implementation of an autonomous hexapod robot with obstacle avoidance and object finding/tracking. The robot has eighteen DOF[1] with each leg having three DOF (degrees of freedom) and the frame is radially symmetric. The mechanical structure is made of an acrylic or Forex frame, steel rods and nineteen servo motors. For the electronic part, it's based on an arduino-mega, placed on the top of the hexapod to manage the six legs in a very easy and efficient manner, and a camera with some sensors used for the robot walking, obstacle avoidance and object finding/tracking. An efficient inverse kinematics engine should be developed to help the robot walk in different ways with obstacle avoidance at variable speeds.

[1] Degrees of freedom

Table of content

Contents

List of figures

List of tables

Introduction

A good scientist is a good mimic. Nature offers an abundant source of knowledge. A fine eye for details and patience is all it takes to unravel its mysteries. There are about one million insect species worldwide. They also probably have the largest biomass of the terrestrial animals with 10 quintillion individual insects alive [1]. In spite of their small size compared to other living creatures and their challenging environment, they proved to be very successful which gives a strong incent to study and mimic them on how they navigate their environment.

Nature doesn't approve of wheels. There are countless creatures that rely on legs for locomotion, ranging from biped (two legs) creatures like us to many-legs creatures (millipedes being the extreme case with leg count between 200 and 750 legs). There are also no natural roads for wheeled vehicles. Despite this fact we rely on wheels for our modern locomotion.

In this work we'll shed some light on the benefits of legged locomotion by designing a six legged robot that is inspired from insects and spiders. The robot is autonomous. It provides two modes of control: user controlled and automated control. The robot can generate a walking gait autonomously with obstacle avoidance. In addition, and with the help of a mounted camera on the robot, the user is able to visualize the environment in which the robot is navigating. According to those built-in abilities, our robot may serve as a vital tool for several systems used in discovering, tracking and surveillance applications.

Report Organization

This work is divided into three chapters. The first chapter focuses on the theoretical background. We'll go through the amazing properties of hexapods and the mathematics behind it. The second chapter deals with the conceptual aspect of the project, including both the mechanical and electronics sides. The third chapter talks about the software that glues everything together. Finally we'll talk about the tests we've run and some guidelines for future improvements.

Chapter 1 : Theoretical background

1.1 Legged Robots: Overview and Classification:

A robot is a machine that can perform a complex task. Robotics is the science behind creating these machines. Robots that were concealed in the realm of science fiction are now gaining more and more importance in our daily lives. The first legged robot in the modern sense is the "Phony Pony" [2] by Mcgee and Frank, University of South Carolina 1968 as shown in figure 1.1. It had four legs with two degrees of freedom in each leg allowing it to move forward and backward only. The walk was programmed with a simple state machine.

Starting from that time, there were a lot of legged robots that can be categorized according to the number of legs as follows:

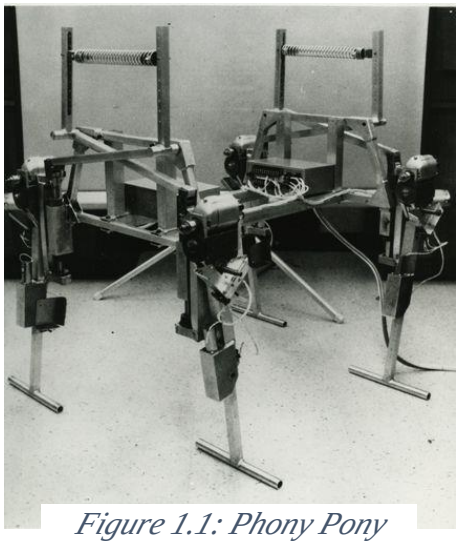

Figure 1.1: Phony Pony

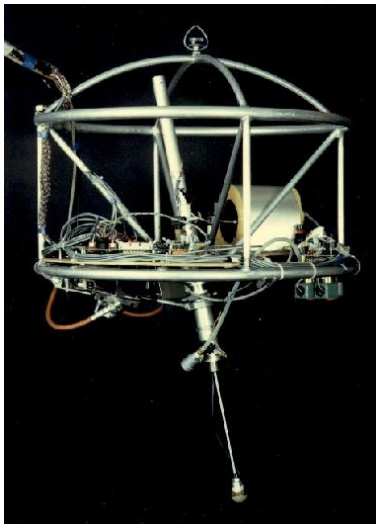

Figure 1.2: One Leg Hopper

1.1.1 The one leg hopper (1983-1984)

The one leg hopper robot [3] is shown in figure 1.2. It relies on dynamic stability in order not to fall. The principle is really simple. The robot hops all the time and with each hop it adjusts its center of gravity by applying a corrective force in order not to fall. The advantage of such robot is its ability to walk in any terrain as it has only one contact point with the ground. It can jump a considerable distance which allows it to avoid obstacles. Having one leg removes the hassle of coordinating the movement of many legs and reduces the energy consumption considerably. The main disadvantage is the complexity of the design and control.

1.1.2 Bipeds (two legged robots or Humanoids)

With two legs those robots are designed to mimic human beings. They're dynamically stable and they can perform pretty much any walk that we humans can do. The disadvantage is the complexity of the design and control. The most notable examples are "ASIMO" by HONDA and "Atlas" by Boston Dynamics.

1.1.3 Tripeds (three legged robots)

With three legs, these robots are statically stable, however they need a complex control algorithm in order not to fall when walking. They didn't receive much attention because there isn't any creatures with three legs in nature. The most notable example is "STriDER" [4] developed at Virginia Tech University 2007. It has a unique and very innovative gait. Despite the fact that it has 3 legs, its gait is closer to the human walk than the biped robots as it allows the knees to swing freely like human's do.

1.1.4 Tetrapods (Four legged)

With four legs they mimic most terrestrial animals. They're statically stable and can move in two main modes. The statically stable mode where only one leg is raised at a time and the dynamically stable mode where 2 or more legs are in the air at the same time. The first mode offers great stability and simplicity of control while the second increases speed in expense of complexity of control and leg coordination. The most notable examples are "Big Dog" (Discontinued), "LS3", "Spot" and "Cheetah" [5] from Boston Dynamics.

1.1.5 Pentapods (five legged robots)

These robots are statically stable. Some of their gaits are inspired from starfish and others are optimally generated using learning algorithms through trial and error as there's no terrestrial walkers with five legs [6].

1.1.6 Hexapods (six legged robots)

Hexapods raised a great interest in both the scientific and hobbyist communities. They have great static stability when standing or walking. Having six legs also offers redundancy in case any leg is damaged and the abundance of six legged creatures allows for biologically inspired gaits to be programmed. One of the first advances in hexapod was the "Adaptive suspension vehicle" (1985) figure 1.3 [7] which was a six legged vehicle that could lift up to 226 KG of payload. There are also many hexapods available freely thanks to the contribution of many great minds most notably

Figure 1.3: Adaptive Suspension Vehicle

“Stubby”, “Phoenix”, “Hexy”. A model worth studying too is “Thex” developed by Boston Dynamics.

1.1.7 Eight legged robots

These are inspired by spiders. They have an amazing stability, they offer redundancy if many legs are damaged and they can walk on any terrain. The drawback is the complexity of leg coordination and the increased cost.

1.2 Why did we chose the Hexapod?

Hexapods are in the middle of the spectrum of legged robots. They are statically stable when standing or walking. There is a large documentation about them available freely. They can walk virtually on any terrain and avoid many kinds of obstacles. With the appropriate gait programming, they can adapt to accidents that may cause some legs to dysfunction which in turns gives them a great advantage over less legged robots. They are energy efficient compared to wheeled vehicles. Hexapods are easier to program compared to eight legged robots and they are considerably cheaper and require less energy.

1.3 Areas of applications

Hexapods do a great work when we need something to be transported over a rough terrain. With the appropriate modifications they can accomplish other tasks like logging figure 1.4. They can be used to roam and discover unknown terrains and even sent to unknown planets, smaller models can be sent into disaster areas in search for survivors. They are also a fun tool to teach people about robotics and dynamics.

Figure 1.4: Hexapod logging

1.4 Kinematics of the model

Our model is based on "Stubby" developed by Mr Wyatt Olson. In what follows, we'll study the leg forward and inverse kinematics as it is very important to generate the gaits later on.

The design is inspired from insect legs. Figure 1.5 clearly shows the similarity between the robot's leg and an ant's leg.

Each leg has three revolute joints giving each leg three degrees of freedom.

In what follows, we'll study the forward and inverse kinematics of the leg.

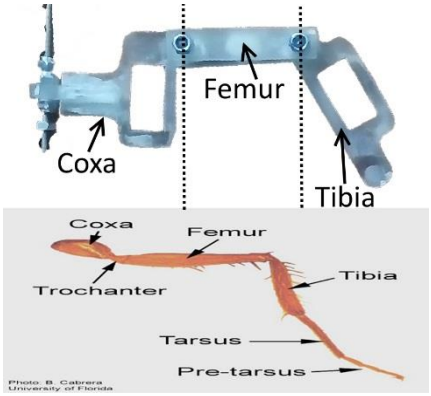

Figure 1.5

1.5 Forward kinematics

Forward kinematics consists of finding the position of the tip of the leg given some specific angles of the joints. This isn't very relevant to our needs, however we need it for diagnosis purposes. . In fact, it is used to verify the results produced by the inverse kinematics function. Figure 1.6 shows the direction for the x, y, and z axis.

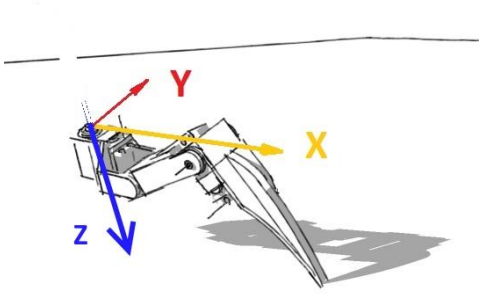

Figure 1.6

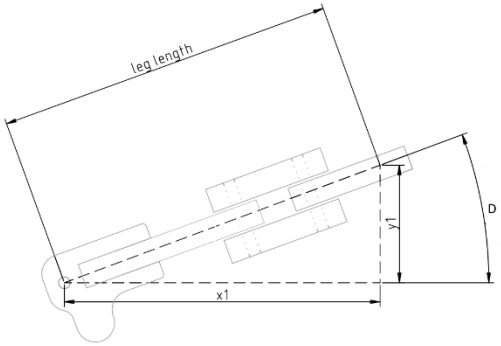

Figure 1.7

To find the end point coordinates, we start from the three joints angles and advance. As we see in figure 1.7 we need the angle ***D*** and the leg length to find ***x*** and ***y***.

We start by finding the angle ***D*** shown in figure 1.8

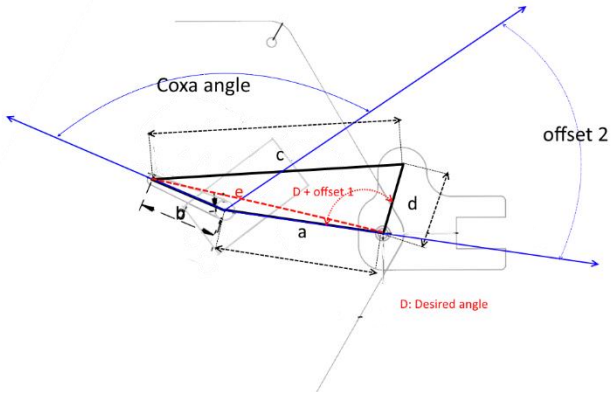

Figure 1.8

We know the lengths ***a***, ***b***, ***c*** and ***d***. We also know the angles ***offset1***, ***offset2*** and ***coxa angle***. We'll use the law of cosines on the triangle ***a***, ***b***, ***e*** to find the length ***e***. Then, we'll use the law of cosines again on the triangle ***a, b***, ***e*** to find the angle between "***a***" and "***e***" let's call it ***C1*** Then we apply the same law on the triangle "***e, d, c"*** to find the angle between "***e***" and "***d***" let's call it ***C2.***

The Desired angle, let's call it ***D***, is then $C1 + C2 - offset1$

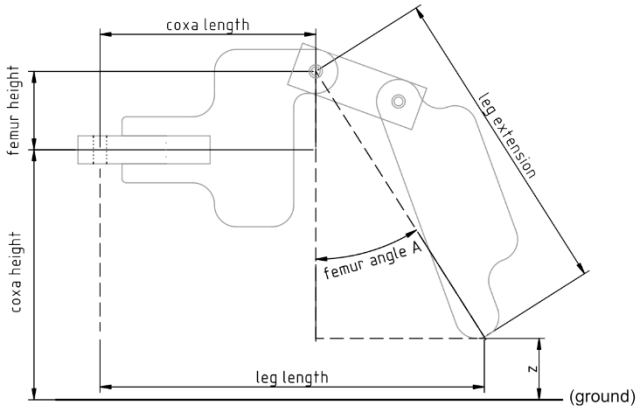

Figure 1.9

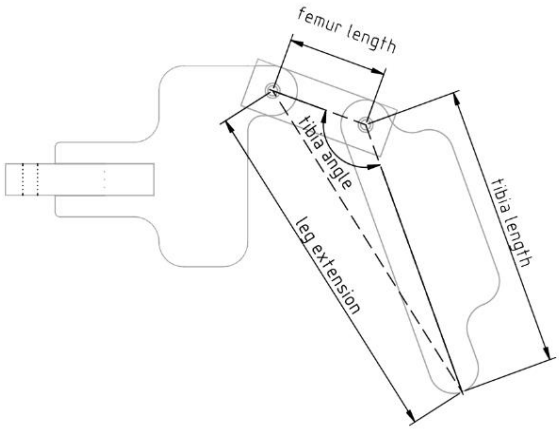

Figure 1.10

To find the Leg length we'll move to the Tibia and femur joints

As we see in the figure 1.9 we need to find the ***leg extension*** to find both ***z*** and ***leg length***.

We calculate the ***leg extension*** from figure 1.10 using the law of cosines.

As we can see we need to find the ***tibia angle*** first, which in turn is calculated from figure 1.11

First we calculate the length ***e*** using the triangle "***a, b***, ***e***"

Then we calculate the two segments of the ***desired angle + E offset*** using the triangles ***"d, e, c"*** and ***"a, b, e"***

After that we get back to figure 1.10 in which we measure ***femur length*** and ***tibia length*** and we use the law of cosines to calculate the ***leg extension***

To find ***z*** and the ***leg length*** we first find ***femur angle A*** which is equal to the desired angle of figure 1.12 minus the ***tibia angle*** of figure 1.10.

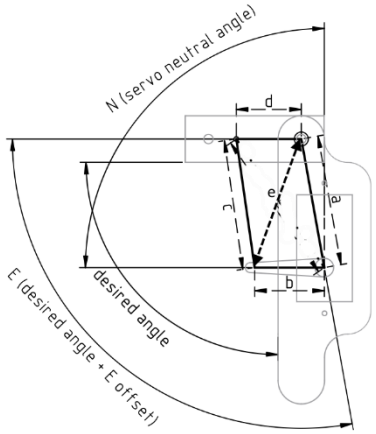

Figure 1.11

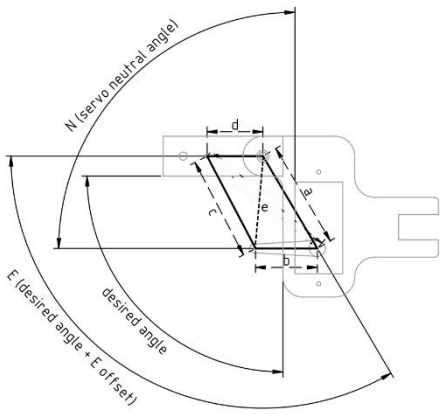

Figure 1.12

Back to figure 1.12

The ***desired angle*** is the angle between ***"a"*** and ***"d"*** minus the ***E offset***

To find the angle between ***"a"*** and ***"d"*** we use the law of cosines to find ***e*** then to calculate both halves of the angle.

Now we get back to Figure 1.9 and compute ***z*** and ***leg length*** using simple trigonometry.

Then we get back to figure 1.7 and calculate both x and y and we're done!

1.6 Inverse kinematics

Inverse kinematics is about finding the joints' angles that would position the leg at a certain point. In other words we have ***(x,y,z)*** and we need to find the angles of the three revolute joints.

Here we start from Figure 1.9 and find the ***leg length*** using Pythagorean Theorem

We then use figure 1.13 to find the ***coxa angle***

Coxa angle = angle between "a and d" – offset2

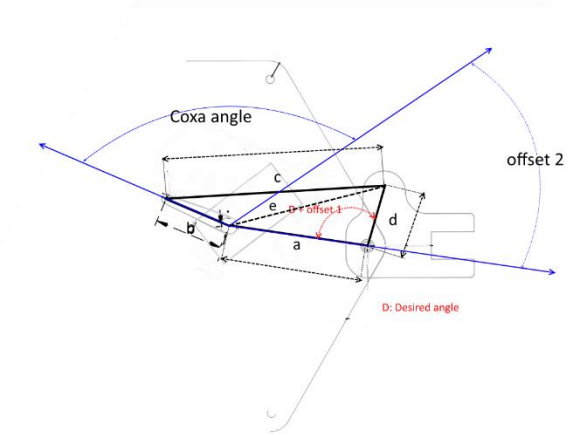

Figure 1.13

Angle between ***"a"*** and ***"d"*** is found using the law of cosines on the two triangles ***"b, e, c"*** and ***"e, a, d"***

Then we use figure 1.9 to find the ***leg extension*** using Pythagorean Theorem. After that we move to figure 1.10 and calculate ***tibia angle*** using the law of cosines. Finally we move to figure 1.14 to find the ***desired tibia angle*** by applying the law of cosines on the triangle ***"e, d, a"*** to find ***e*** then apply it once again on the same triangle and triangle ***"e, b, c"*** to find ***E(desired angle + E offset).*** We then reduce the offset to find the ***desired tibia angle***

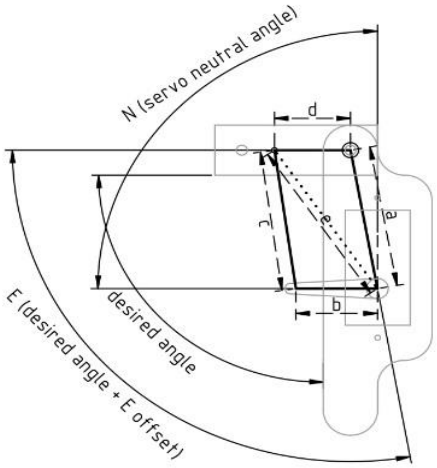

Figure 1.14

The femur angle is divided into two angles: ***femur angle A*** as seen in figure 1.9 and ***femur angle B*** as seen in figure 1.15.

Femur angle B is computed using the law of cosines from the triangle ***femur length, leg extension, tibia length*** in figure 1.15

For ***femur angle A*** we use the Pythagorean Theorem and some simple trigonometry to find it from figure 1.9

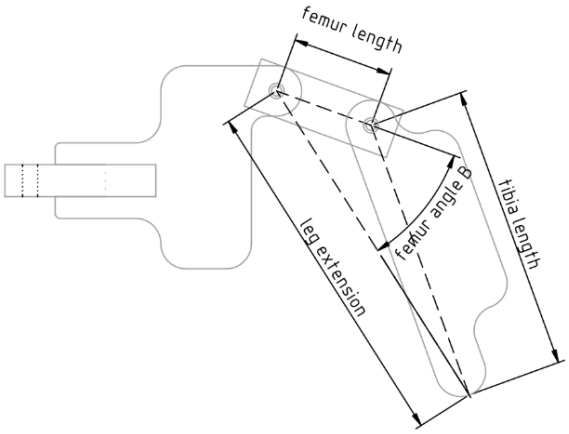

Figure 1.15

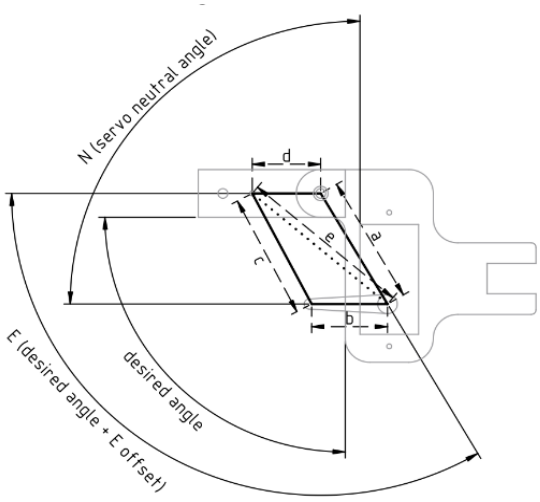

Figure 1.16

Finally we move to figure 1.16

$$\boldsymbol{The\ desired\ femur\ angle\ =\ femur\ angle\ A\ +\ femur\ angle\ B}$$

Another way to compute the femur angle is to apply the law of cosines on the triangles ***"a, d, e"*** and ***"e, b, c"*** to find the angle between ***"a"*** and ***"b".*** We then add the ***offset angle*** that we can measure to find the ***desired coxa angle***

And we're done!

1.7 Gait generation:

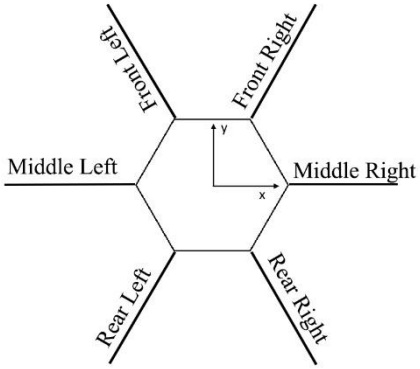

Figure 1.17

A gait is the pattern in which the legs move. It can be random or periodic.

For all what follows, we'll use the notation for the legs shown in figure 1.17.

For simplicity, we'll assume that all the legs are mounted at the center of the robot which coincides with the ***x, y*** axis shown in figure 1.17 .The middle right leg is mounted at angle 0° and we go anticlockwise adding 60° on each leg.All the work is done on the middle right leg and then through the magic of mathematics (translation & rotation matrices) we expand the results to the other legs.

A single leg performs a step by following some predefined points. When the points belonged to cosine function curve there were a bit of inconsistency when the legs contact the ground. To solve this problem, we used Bezier curves [8] instead of cosine function. The iterative procedure for generating Bezier curves is shown in figure 18. After experimentation we changed the pattern to get the best results.

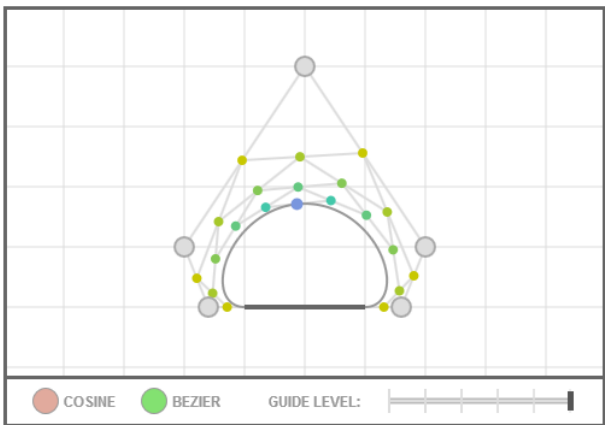

Figure 1.18

In summary, the step works as follows:

The leg moves backwards a little bit, then it raises to the air and advancing forward. It then contacts the ground and goes back to the starting position while pulling the body forward.

Now that we're done with the step, we can generate the gaits.

The number of gaits depends on the number of legs. For instance if k is the number of legs and N is the number of gaits we have: $N = (2k - 1)!$ [9]

Our robot has 6 legs which gives rise to$11! = 39916800\ gaits$

Each leg has 3 DOF meaning that the robot has 18 DOF. This gives rise to even more redundancy in the gaits.

It would be tiresome even for a super computer to go through all of them. Thankfully we've got nature to turn to for inspiration.

There are two important gaits that we will study: the wave and tripod gait.

Before we dive into the gaits, there are two rules that we need to obey to ensure the robot's stability.

- First rule: never have two neighbouring legs raised from the ground at the same time.
- Second rule: No leg should be allowed to perform a step unless it obeys rule one and its neighbouring legs stepped recently.

1.7.1 The wave gait:

The name comes from the fact that the leg movement creates two waves on each side of the hexapod. This is the most stable gait which makes it very suitable for fragile terrains. The only drawback is the speed.

To generate this gait, we add a simple rule to the previous ones: only one leg is in the air at any time. The sequence is demonstrated on table 1-1 where shaded blocks signify leg is on the air.

Table 1-1: Wave gait sequence

Time →								
Front Right								
Middle Right								
Rear Right								
Front Left								
Middle Left								
Rear Left								

1.7.2 Tripod gait:

We generate this gait by obeying the first two rules and raising three legs at a time. This gait is the fastest statically stable gait. At any time the body is supported by three legs while the other three are in the air. Table 1-2 demonstrates the sequence.

Table 1-2: Tripod gait sequence

Time →								
Front Right								
Middle Right								
Rear Right								
Front Left								
Middle Left								
Rear Left								

Chapter 2 : Hardware System Design

2.1 The Robot Frame:

The design of the mechanical body was taken from “Stubby” made by Mr Wyatt Olson. We’ve taken the permission to reproduce his work at will.

We’ve made two bodies for the robot with two different materials. The parts were manually cut and refined. The first one was made using Plexiglas 6mm thick. The first robot including the batteries was about 700g. The second was made using Forex 5mm thick which is significantly lighter but less resistant. It weighted about 595g which is a significant reduction in the overall weight. (The camera and raspberry Pi weren’t included in the weighting)

Figure 2.1 is a picture of the mounted robot. Figure 2.2 shows all the mechanical parts with their dimensions

Remark: we have six legs, so the leg section is repeated six times.

Furthermore, to construct the frame we needed the following components and tools:

1- 8x 70mm lengths of M4 threaded rod
2- 12x 32mm M4 rods
3- 28x M4 Nuts
4- 46x M4 Lock Nuts
5- Cutter with multiple blades
6- Scroll saw
7- Many files with different
8- Sandpaper
9- Glue
10- Rigid wire of 1mm (or less) diameter
11- Patience (lot of it)

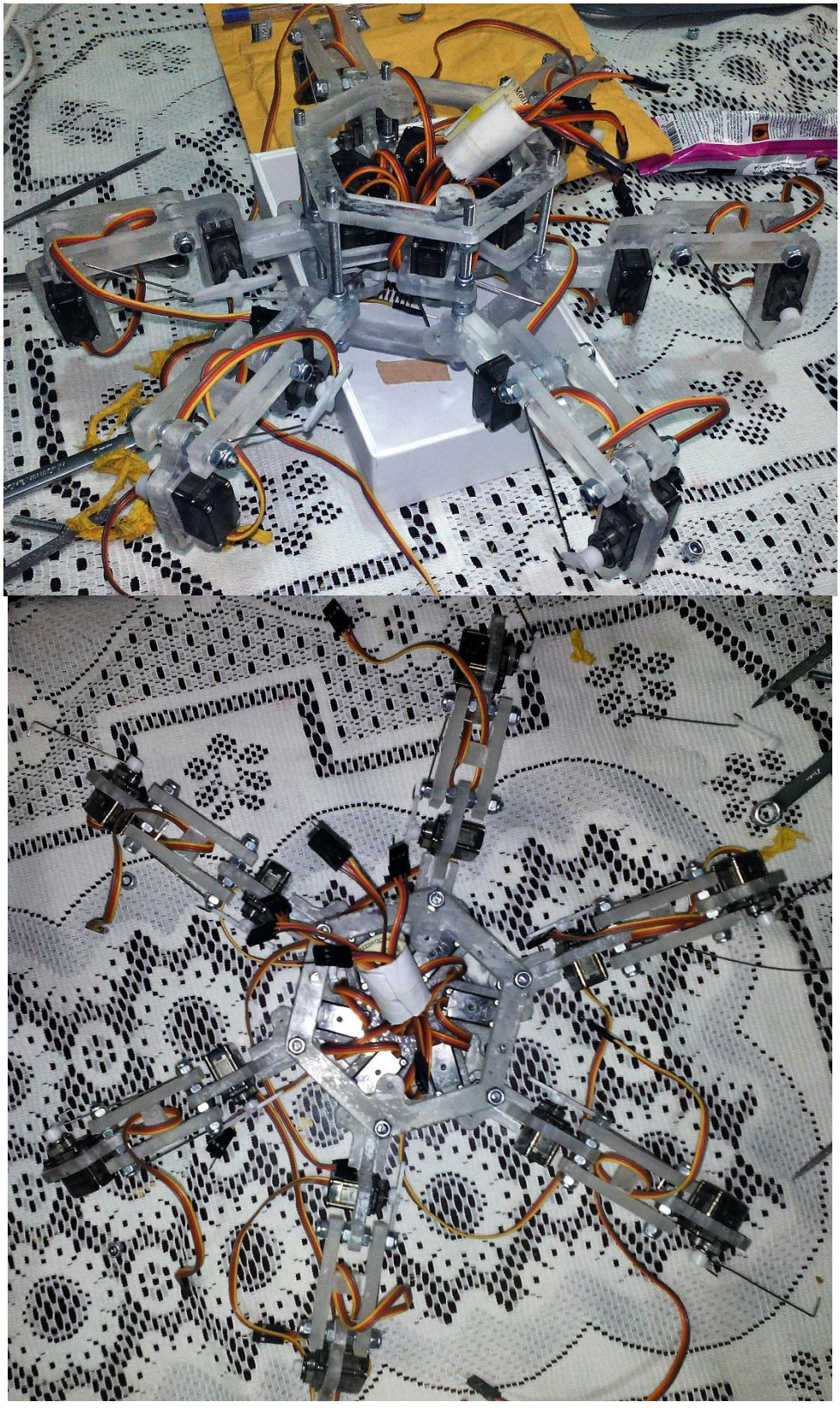

Figure 2.1: Mounted Robot

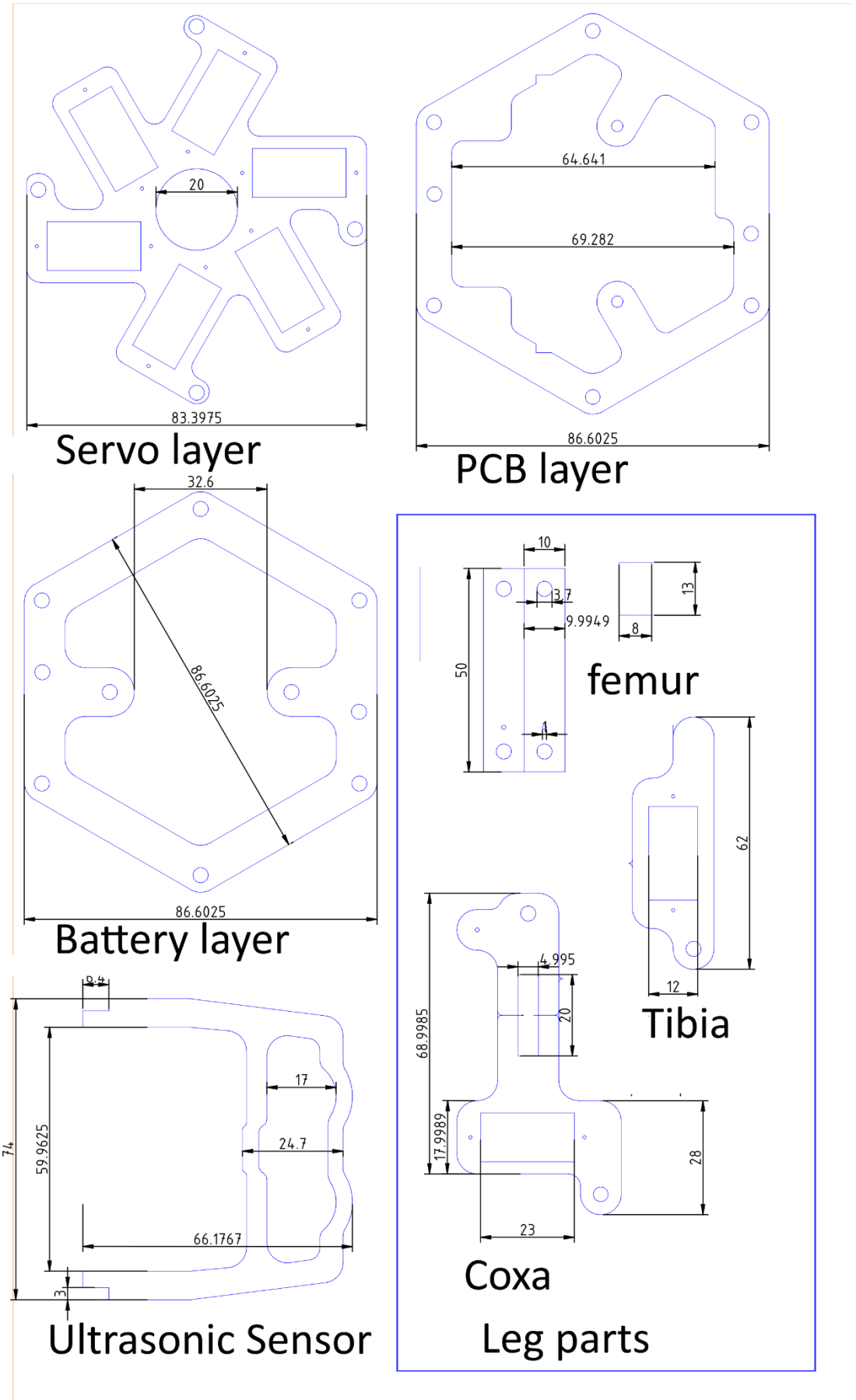

Figure 2.2: Frame Parts

2.1.1 Assembling the Frame

2.1.1.1 Leg assembling

Figure 2.3: Coxa Assembled

First we glue the two coxa parts as shown in figure 2.3. Then we glue the femur parts together. After that we attach the tibia to the femur using a 32mm rod. The tibia is held tightly using two nuts and the rod is secured using two lock nuts. Using the same technique we attach the coxa to the femur and finally we add the servo motors. It's better to use glue to hold the servos in their places. We then place the pushrods that connect the servo motors with the parts they control. The pushrods lengths shouldn't necessary match the ones we used, however all the pushrods that control the same parts should be equal figure 2.4.1.

For our design we used the following lengths:

Tibia rod = 55 mm

Femur rod = 40 mm

Coxa rod = 35 mm

We then screw a locknut 21mm into the 70mm rod figure 2.4.2, we then place the rod into the coxa with the smaller end pointing to the ground figure 2.4.3 and we secure the rod using another locknut figure 2.4.4.

Two legs should have one extra locknut on the 70mm rod on the side pointing up and one regular nut on the side pointing to the ground. They would serve as space holders.

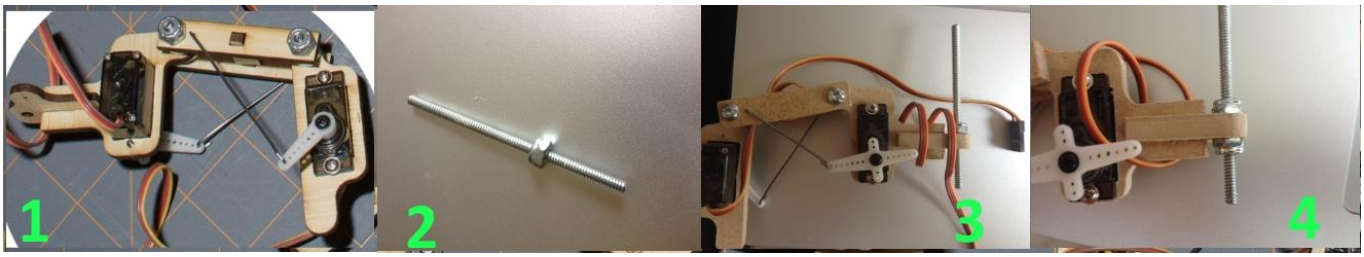

Figure 2.4: Leg Assembly

2.1.1.2 Body assembly

- Place the servo motors on the servo layer as in figure 2.5.
- Place the two legs with the space holders in the positions indicated in figure 2.6 and hold them using lock nuts.
- Add the other legs and place the servo motors layer.
- Place locknuts over them.
- Carefully attach the Coxa push rods.
- Then add locknuts on the spacers leaving 12mm to place the pcb layer.
- Place the PCB layer
- Finally add locknuts to hold everything together.

Figure 2.5: Servo Layer

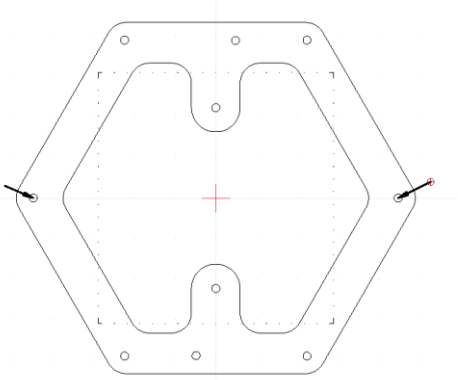

Figure 2.6: Space holders

2.2 Electrical components

The various electrical components included in our design are as follows:

- 18x 9g micro servo motors.
- Arduino mega
- Ultrasonic Sensor HC-SR04
- 2x 10000 µf capacitors
- Wi-Fi Camera
- Gamepad
- Computer with Wi-Fi capability
- 5V – 40A Power Supply
- 4x rechargeable AA NiMh batteries with capacity of 1900mAh or more

2.2.1 Micro servos:

Figure 2.7: Micro Servo

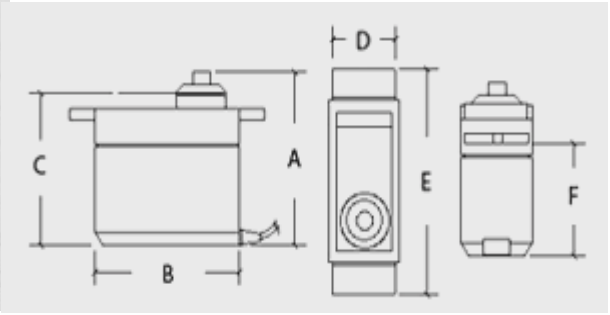

Figure 2.8

The micro servo shown in figure 2.7 is one of the micro servos used in our design. This is because they're light (12g with the wires), and they're cheap about 3$.

The disadvantage is their sensibility to noise (they can shake in some angles and there's little to do about it) and relatively low torque.

Geometrical and electrical specifications of the servo are gathered in figure 2.8 and table 2-1.

Table 2-1: Servo specifications

Weight (g)	**9**
Torque (kg)	1.5kg/cm (4.8V), 1.7kg/cm (6.0V)
Speed(Sec/60deg)	0.13sec/60° (4.8V), 0.12sec/60° (6.0V)
Operating voltage	4.8~6.0V
Operating current	80mA(4.8V) 100mA(6V)
Stall current	500mA(4.8) 650mA(6V)
A(mm)	29
B(mm)	23
C(mm)	25
D(mm)	12
E(mm)	32
F(mm)	19

2.2.2 Arduino Mega

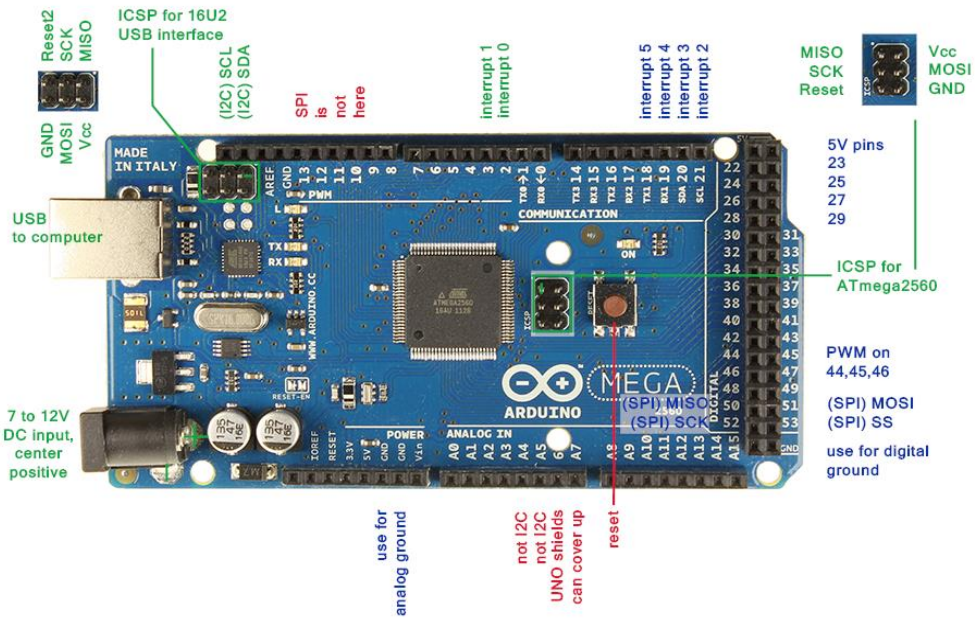

Figure 2.9:Arduino Mega

The Arduino mega figure 2.9 uses the microcontroller ATmega2560. The arduino controls the 18 servos, receives data from the Ultrasonic sensor and commands from the gamepad. The full specifications are on table 2-2.

Table 2-2: Arduino mega specifications

Microcontroller	ATmega2560	DC Current for 3.3V Pin	50 mA
Digital I/O Pins	54 (of which 15 provide PWM output)	Flash Memory	256 KB of which 8 KB used by bootloader
Input Voltage (recommended)	7-12V	SRAM	8 KB
Input Voltage (limit)	6-20V	EEPROM	4 KB
		Clock Speed	16 MHz
Analog Input Pins	16	Length	101.52 mm
DC Current per I/O Pin	20 mA	Width	53.3 mm
Weight	37 g	Operating Voltage	5V

2.2.3Ultrasonic Sensor (HC-SR04)

The Ultrasonic sensor figure 2.10 is a transceiver that sends and receives ultrasound waves and transforms them into electrical signals. The device measures distances by a simple concept. The device sends an ultrasound wave, when it hits an obstacle it bounces back and the device receives it. It is then converted to an electrical signal which in turn is translated into a distance. Our robot uses this sensor to detect obstacles in front of him.

Figure 2.10: HC-SR04

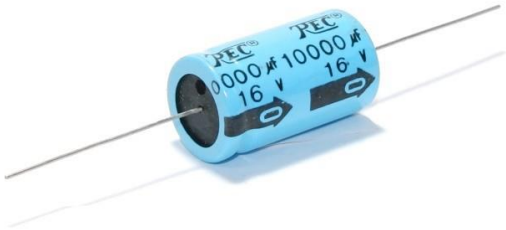

Figure 2.11: 10000uF capacitor

2.2.4 10000µF capacitors

Capacitors figure 2.11"are used to prevent the voltage from dropping when all the servos draw current at the same time. The larger the capacity the better. 10000µF are the largest we could find and we used only two due to weight and space limitations.

2.2.5 Wi-Fi Camera (SJ7000)

Figure 2.12: SJ7000

We chose this camera figure 2.12 for many reasons: first its lightweight (71g) and its small dimensions (59.27 x 41.13 x 29.28mm) makes it suitable for our small hexapod. It doesn't add much to the weight. The other advantage is the Wi-Fi connection. The camera can stream video over Wi-Fi which makes it excellent to use in our robot. The other advantage is that it can be paired with laptops and smart phones too.

2.2.6 Power Supply

The power supply is used for the tethered mode. When the robot is in a controlled environment or when we need to use it for a prolonged period of time (more than 45 minutes), it's suitable to plug the power supply with a long extension cord.

2.2.7 Rechargeable NiMh AA batteries

The choice of batteries was made because of two reasons:

1- Our micro servos need a voltage between 4.8V to 6V, 4x AA batteries in series give 4.8V which eliminates the need for voltage regulation from the circuit.

2- NiMh batteries have a low internal resistance. This property allow them to source enough current for the servos.

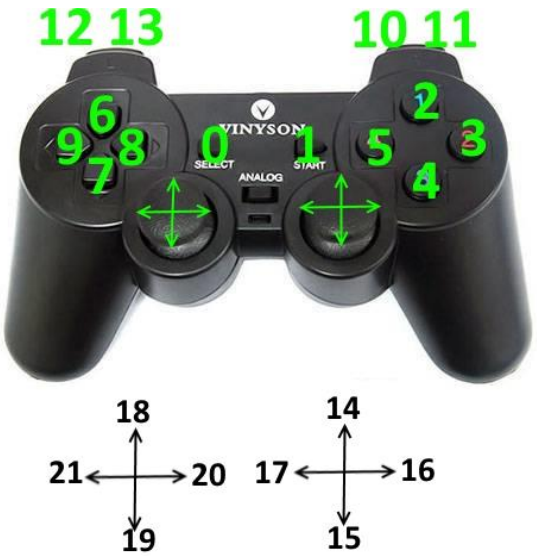

Figure 2.13: Gamepad numbering

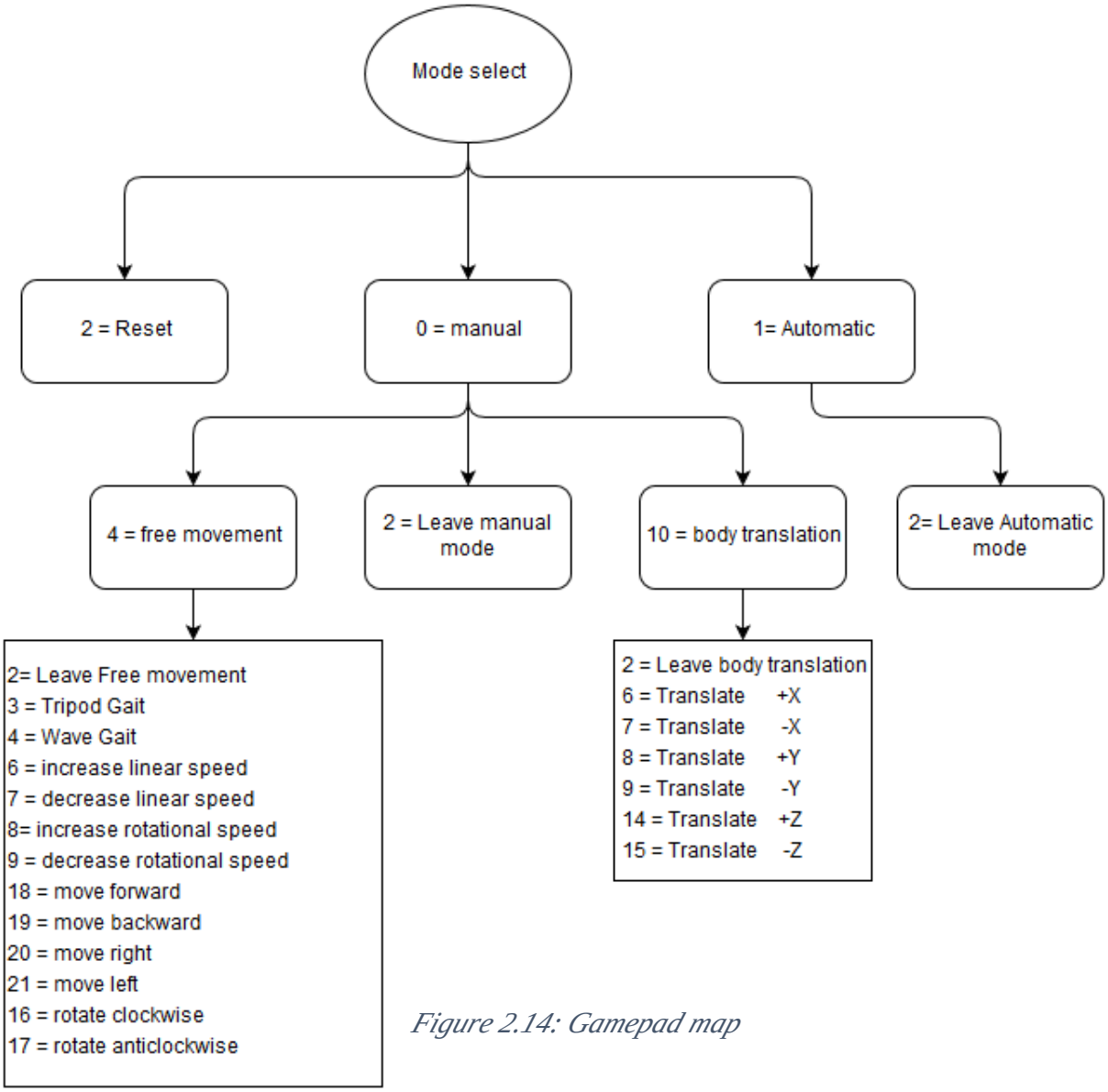

Figure 2.14: Gamepad map

2.2.7 Gamepad

The gamepad figure 2.13 is used to control the hexapod. The instructions are sent to the laptop which in turn sends them to the Arduino using the USB serial communication. The gamepad has 21 buttons. The combinations are explained in the flowchart in figure 2.14.

2.3 The overall System

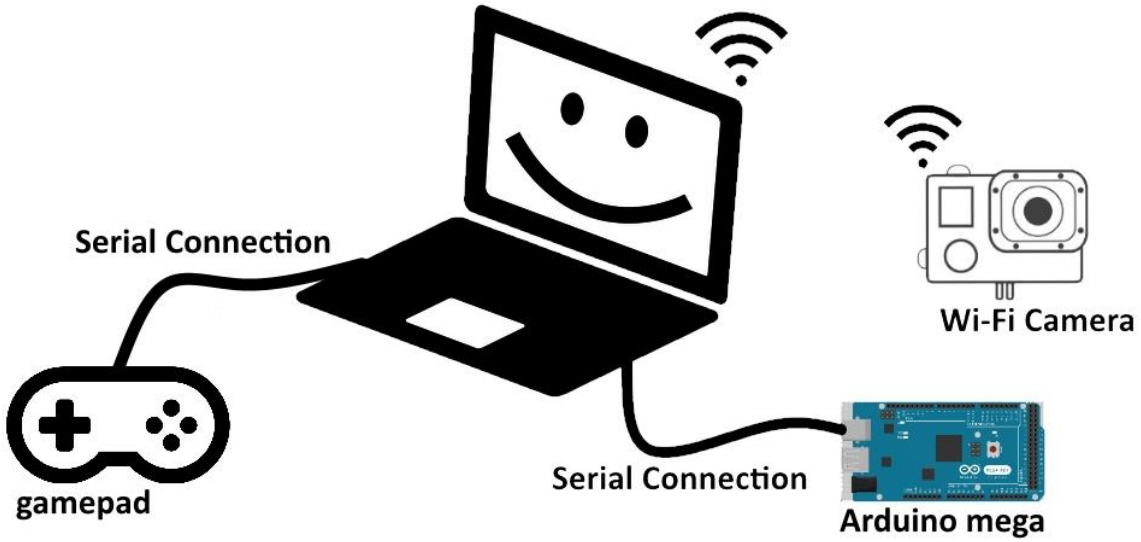

Figure 2.15: connections

To make things clear for the reader, we present two diagrams. Figure 2.15 represents the connections between the laptop, Arduino, gamepad and camera. Originally we planned to use wireless connection between the arduino, the laptop and the gamepad and completely eliminate wires. Sadly the Wi-Fi modules got lost somewhere from China to Algeria. So we had to rely on wired, serial connections.We also wanted to give the robot a sense of direction by detecting the magnetic poles of the earth using a magnetometer. The magnetometer lost its sense of direction and got lost in the transit too. Figure 2.16 shows the connections between the Arduino, servos, batteries, capacitors and ultrasonic Sensor. Table 2-3 shows the Arduino pin assignment for the servos.

Table 2-3: Arduino pins assignment

Leg	Middle Right	Front Right	Front Left	Middle Left	Rear Left	Rear Right
Tibia	36	42	37	43	49	30
Femur	34	40	35	41	51	28
Coxa	32	38	33	39	53	26

Similarly, For the Ultrasonic sensor the following pins of the Arduino are used:

Trigger pin: A6	**Echo pin: A7**

For the servo motor controlling the Ultrasonic sensor: Control pin : 47

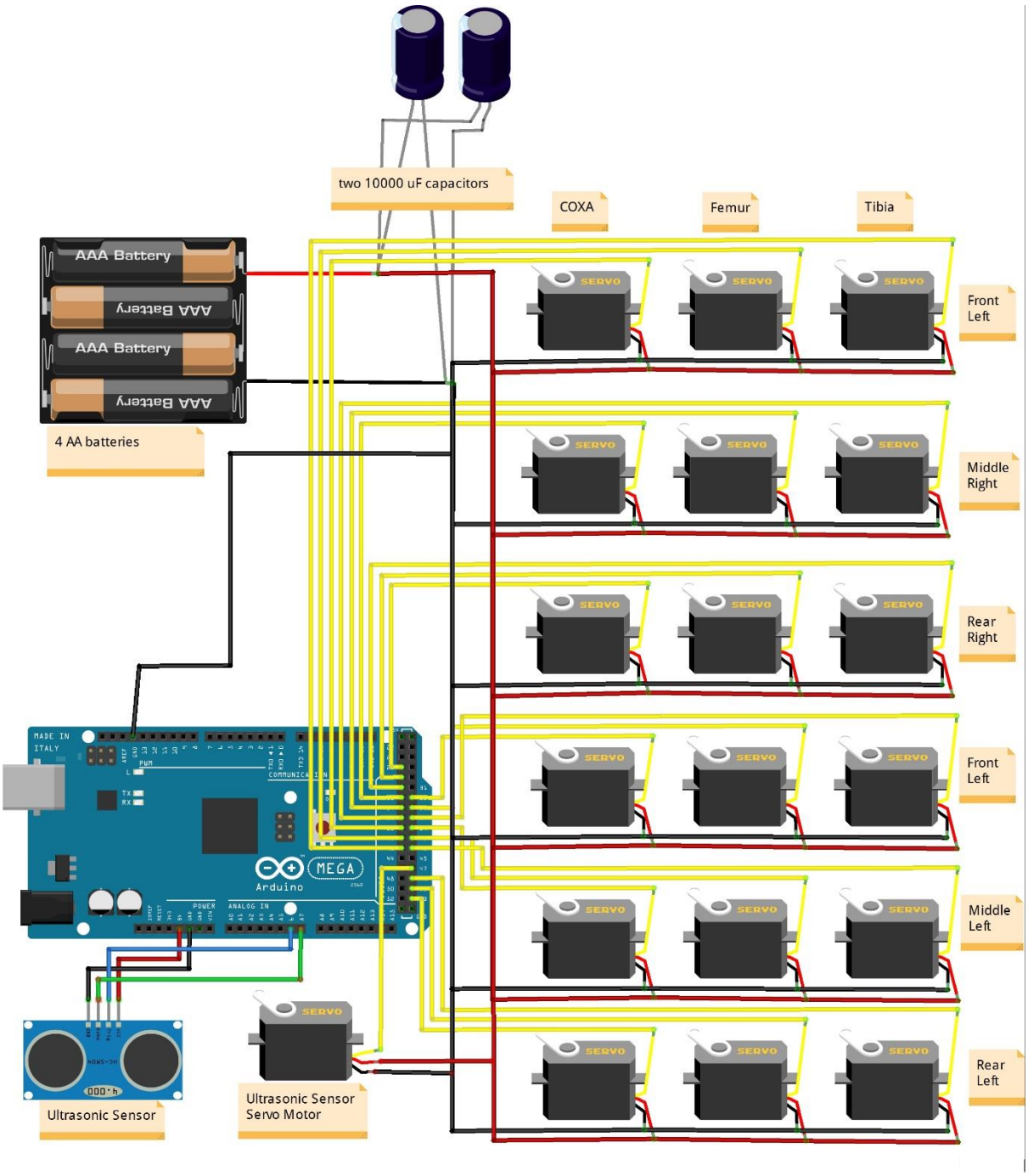

Figure 2.16: Internal Circuit

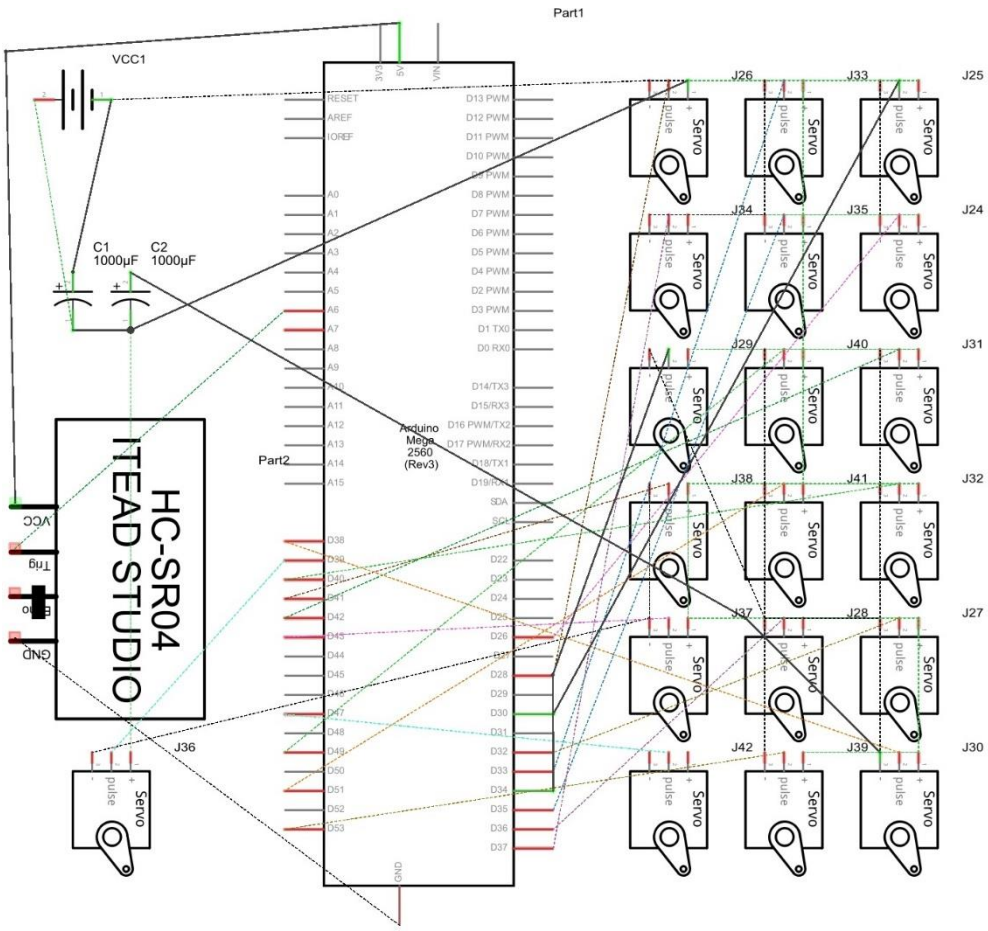

Figure 2.17: Electronic Circuit

Chapter 3 : Software System Design

The software part of our system includes two main programs. A routine that runs on the Arduino and a program that runs on the computer.

3.1 Arduino Side

This piece of code is responsible for the forward and inverse kinematics calculations, gait generation, receiving commands through serial USB and sending information to the computer.

To make system elements more reusable, and thus improving system quality, an object-oriented approach has been adopted in the design of our software. This approach makes use of several classes which are described in the following subsections. Furthermore, the modelling of the system is based on UML (Unified Modeling Language) tool which more suitable for this approach.

3.1.1 Class Point:

This class have three variables representing the position of the point in a 3D space (x,y,z)

The class has the following membership functions:

- A constructor ***Point (int x, int y, int z)***
- ***add (Point offset)*** adds the offset to the point.
- ***set (int x, int y, int z)*** sets the point to the given position.
- ***Rotate2D(float angle)*** rotates a point in 2D space.
- ***rotateXY (float angle)*** rotates a point in the XY plan.
- ***rotateXZ (float angle)*** rotates a point in the XZ plane.
- ***rotateYZ (float angle)*** rotates a point in the YZ plane.

The rotation functions would prove very helpful when it comes to generalizing the results that we get for a single leg to the other ones.

3.1.1.1Derivation of the new coordinates after rotation

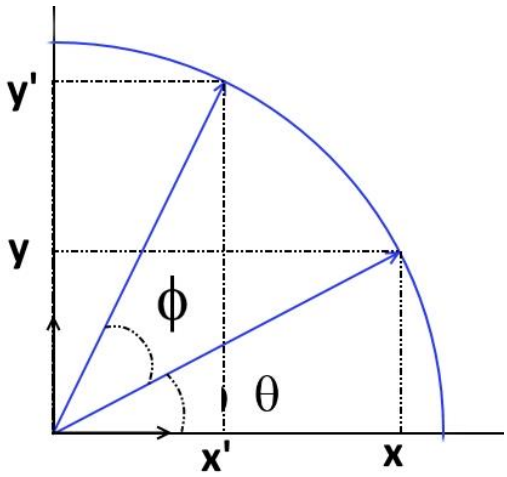

Figure 3.1 Derivation of coordinates

To explain how this is done, we avoided the matrices to make it understandable to anyone based on some basic mathematical geometry knowledge. Refer to figure 3.1.

$$Old\ coordinates\ x, y. New\ coordinates\ x', y'$$

$$\theta = intial\ angle\ ;\ \emptyset = angle\ of\ rotation$$

$$\mathrm{x} = \mathrm{r}\ \times \cos\theta$$

$$\mathrm{y} = \mathrm{r}\ \times \sin\theta$$

$$x' = \mathrm{r} \times \cos(\theta + \emptyset) = \mathrm{r}\ \times \cos\theta \times \cos\emptyset - r \times \sin\theta \times \sin\emptyset$$

$$y' = \mathrm{r} \times \sin(\theta + \emptyset) = r \times \sin\theta \times \cos\emptyset + r \times \cos\theta \times \sin\emptyset$$

$$x' = x\ \times \cos\emptyset - y\ \times \sin\emptyset$$

$$y' = y\ \times \cos\emptyset + x\ \times \sin\emptyset$$

3.1.2 Class Servo

This class was taken from the Arduino library. It offers many useful functions to work with servo motors. We used two functions:

- ***attach (int pin)*** which instruct the Arduino that a servo is attached at the specified pin
- ***write (int angle)*** which instructs the servo to move to the specified angle.

It's worth to describe how does this function works.

The servos are analog devices and we're controlling them using digital outputs from the Arduino. This is accomplished by using Pulse width modulation (PWM). By changing the duty cycle of the pulse we'll get different angles in the Servo. The PWM pulses are generated using the different timers of the ATMEGA 2560. The servos need to be updated each 20ms. This allows using one timer to control many servos.

In the arduino mega we can control up to 48 servo motors. There's only a small limitation on the pins that we can use which is summarized in table 3-1.

Table 3-1: Pins constraints

Servos	**analogWrite Pins**	**Timers used**
1 - 12	not pins 44,45,46	Timer 5
13 - 24	not pins 11,12,44,45,46	Timers 1 & 5
24 - 36	not pins 6,7,8,11,12,44,45,46	Timers 1,4 & 5
37 - 48	not pins 2,3,5,6,7,8,11,12,44,45,46	Timers 1,3,4 & 5

3.1.3 Class Newping (Ultrasonic Sensor)

This class was developed by "Tim Eckel" and distributed as a library along the standard arduino libraries. It contains many functions to facilitate the use of ultrasonic sensors. We only needed two functions:

- constructor ***NewPing (int trigger_pin,int echo_pin,int max_distance)***
- ***ping_cm()*** returns the distance in cm
- ***sonar.ping_median(int iterations)*** Do multiple pings (default=5), discard out of range pings and return median in microseconds
- ***sonar.convert_cm(int echoTime)*** Converts microseconds to distance in centimeters

3.1.4 Class Serial

This class is derived from a standard library in the Arduino. It handles the serial communication. We used the following functions:

- ***begin (int baudrate)*** sets the data rate in bits per second
- ***available()*** returns the number of bytes available for reading
- ***read () : char*** returns the first byte of the incoming serial.
- ***println (String string)*** Prints data to the serial port as human-readable ASCII text

3.1.5 Class Leg

This is the most important class in the project. It encapsulates all the physical aspects of the leg (as seen in the class diagram), functions responsible for calculating the forward and inverse kinematics, calibration functions and functions to set the servos to the appropriate angles.

The core of this Class is *"setPosition (Point point)"* function figure 3.2. It takes as input the desired point where we want the end of the leg to be in and does the inverse kinematics to get the required servo angles. After that it sets the servos to the appropriate position.

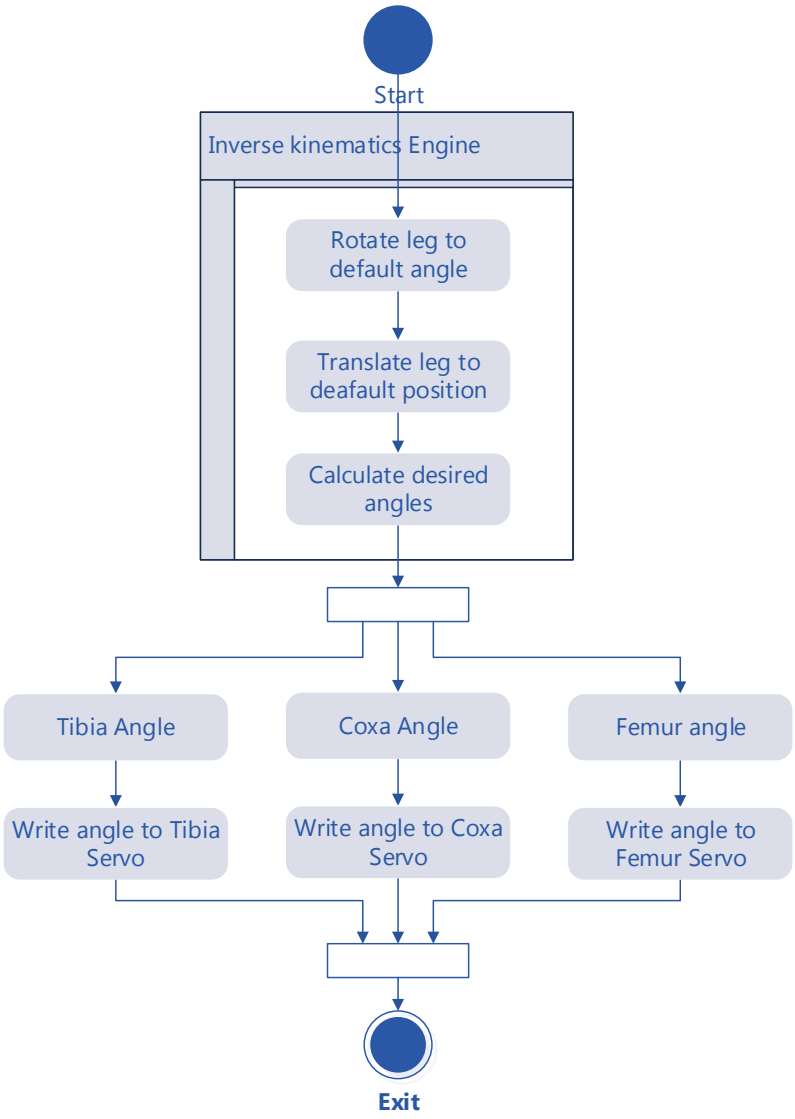

Figure 3.2: setPosition function

3.1.6 Class Rusty

The class is named after the robot. It handles the communication of the arduino with the computer. It also handles, calibration, the step generation, gait generation and the synchronisation between the legs.

Class Diagram in figure 3.3 shows the relation between the classes.

3.1.7 UML Class Diagram

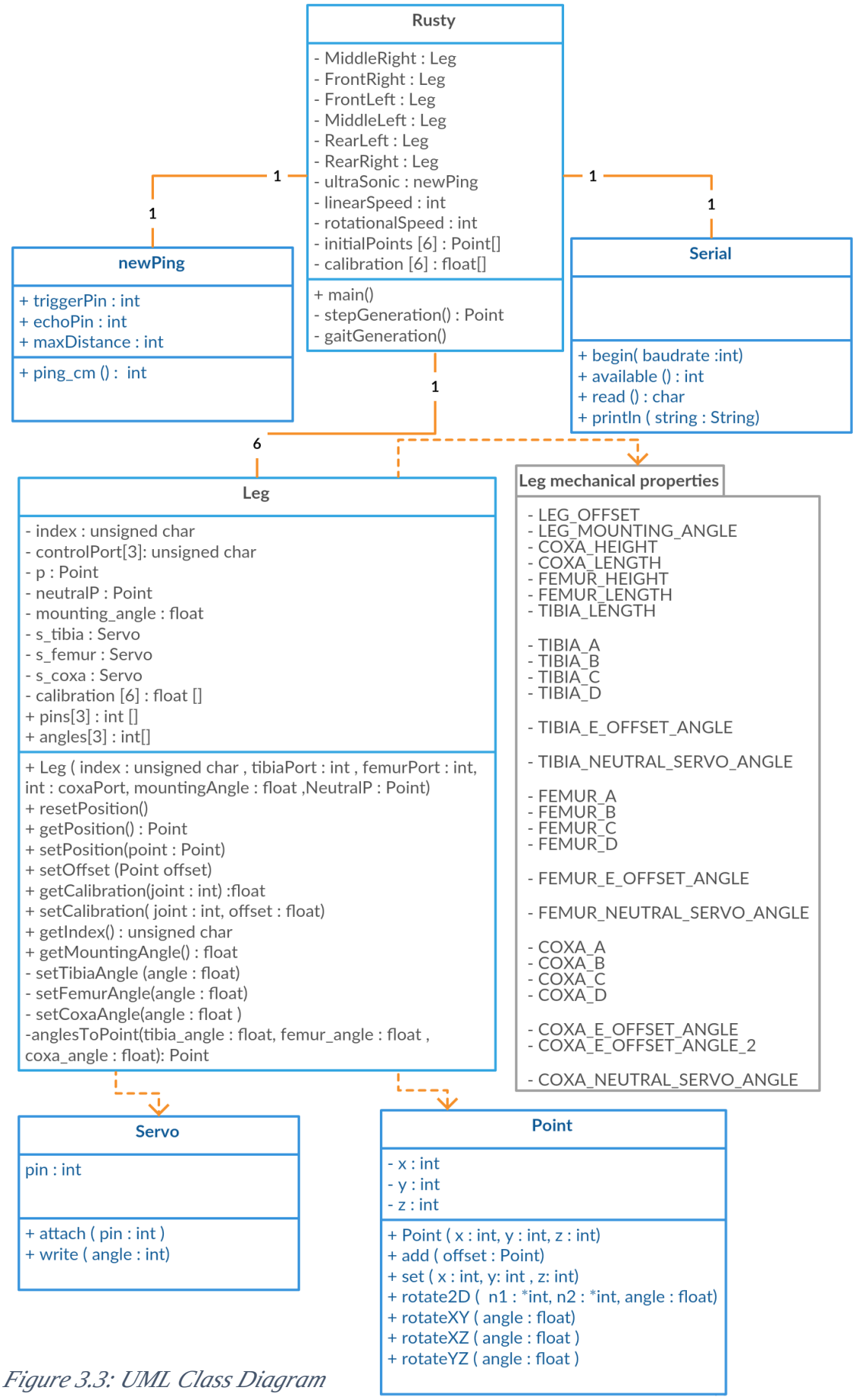

Figure 3.3: UML Class Diagram

3.2 Computer Side

In the computer we have four independent programs running in parallel.

- The first program translates the commands from the gamepad into characters in order to send them through serial connection to the Arduino
- The second program handles the actual serial connection with the Arduino and ensures the synchronization.
- The third program handles the communication with the camera over Wi-Fi. It converts the video stream into a format that is accepted by the program doing the image processing
- The fourth program handles image processing.

3.3 Description of the system

3.3.1 System Start-up

1- The robot s turned on by:
    1- Turning on the Arduino mega this would automatically power the ultrasonic sensor.
    2- Supplying the servos with power.
    3- Turning on the camera.
    4- Establishing the link with the computer.
2- On the computer side:
    1- Plug the gamepad.
    2- Start the software that translates the key presses.
    3- Start the software handling the serial communication.
    4- Establish the Wi-Fi connection between the camera and the computer.
    5- Now depending on the use of the camera we can either:
        1- Start the software that streams the video from the camera.
        2- Start the software that does the image processing.

3.3.2 Use of the system

The robot boots and sets the servos at the default position and waits for orders from the user.

The use then choses the free mode or the automated mode.

The first mode lets the user control the robot. For instance he can make the robot walk four direction, rotate clockwise or anticlockwise and perform body translations on x,y and z axes.

All the movements are done autonomously in the sense that the gait generation and the legs manipulation are performed by the robot itself.

Body translation is changing the position of the body with respect to all supporting legs

The second mode sets the robot free to explore its environment using the feedback from the ultrasonic sensor to detect and avoid obstacles while giving the user a live view using the camera. The robot can also look for an object while in motion. Once it detects the object it would circle it with a green circle. When the robot gets closer to the desired object, the circle turns red and the robot pauses declaring that it did find the object.

3.3.3 UML Activity diagrams

The next activity diagrams explain in a concise way the working of the robot.

- Figure 3.4 shows the start menu of the robot.
- Figure 3.5 shows the body translations.
- Figure 3.6 shows the free movements.
- Figure 3.7 explains the things that happen on the computer.
- Finally figure 3.8 describes the free roaming algorithm.

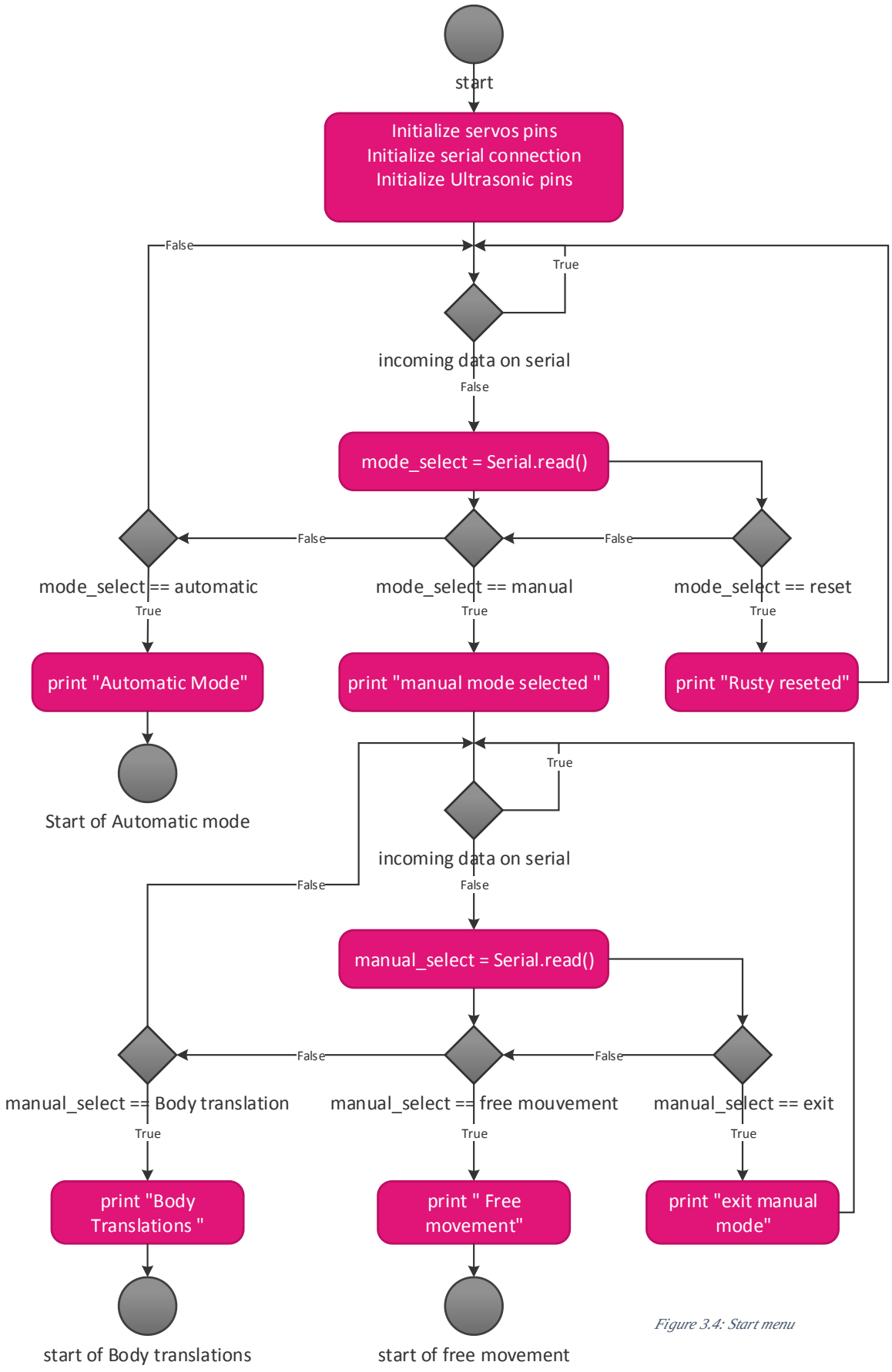

Figure 3.4: Start menu

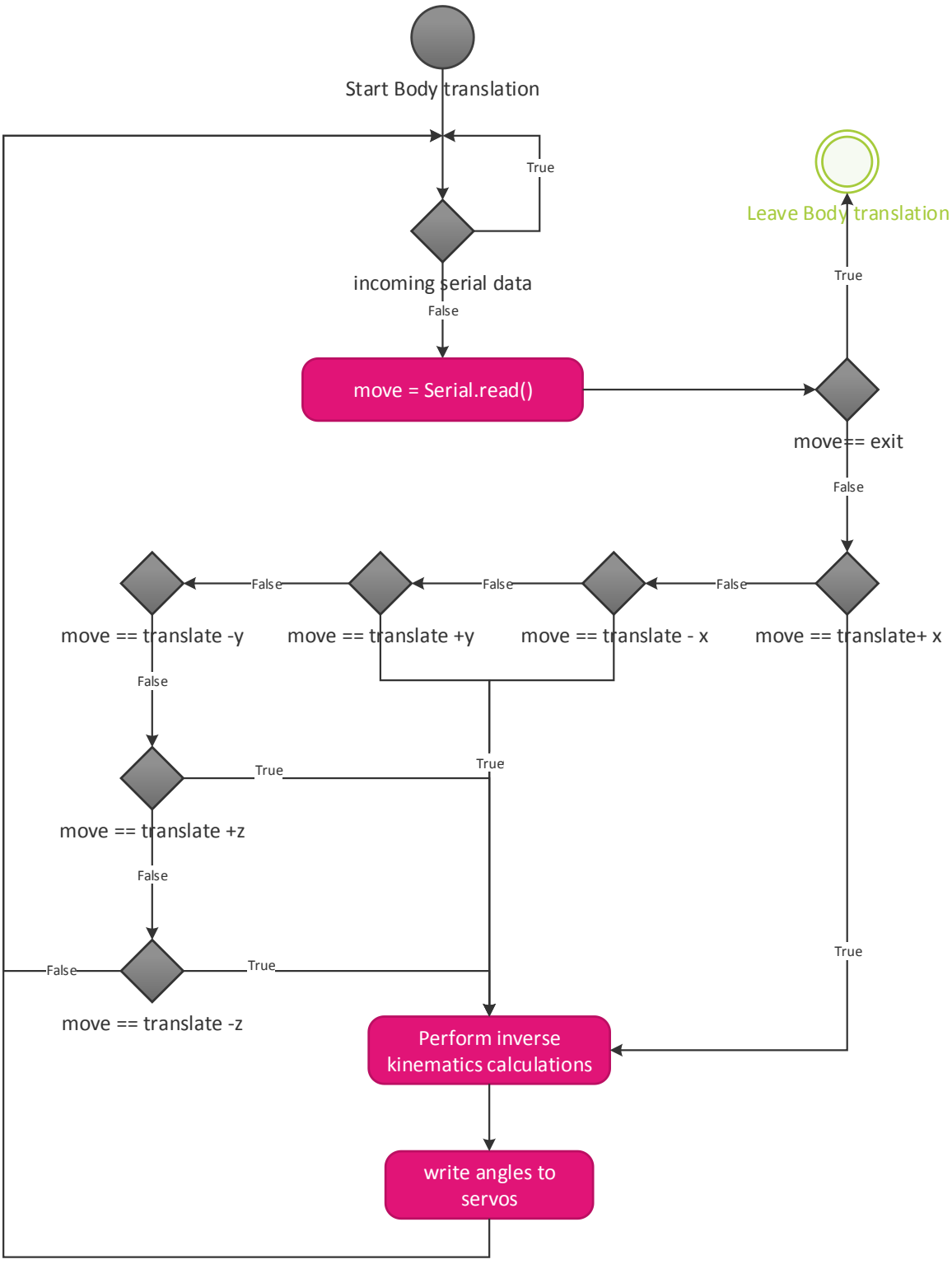

Figure 3.5: Body Translations

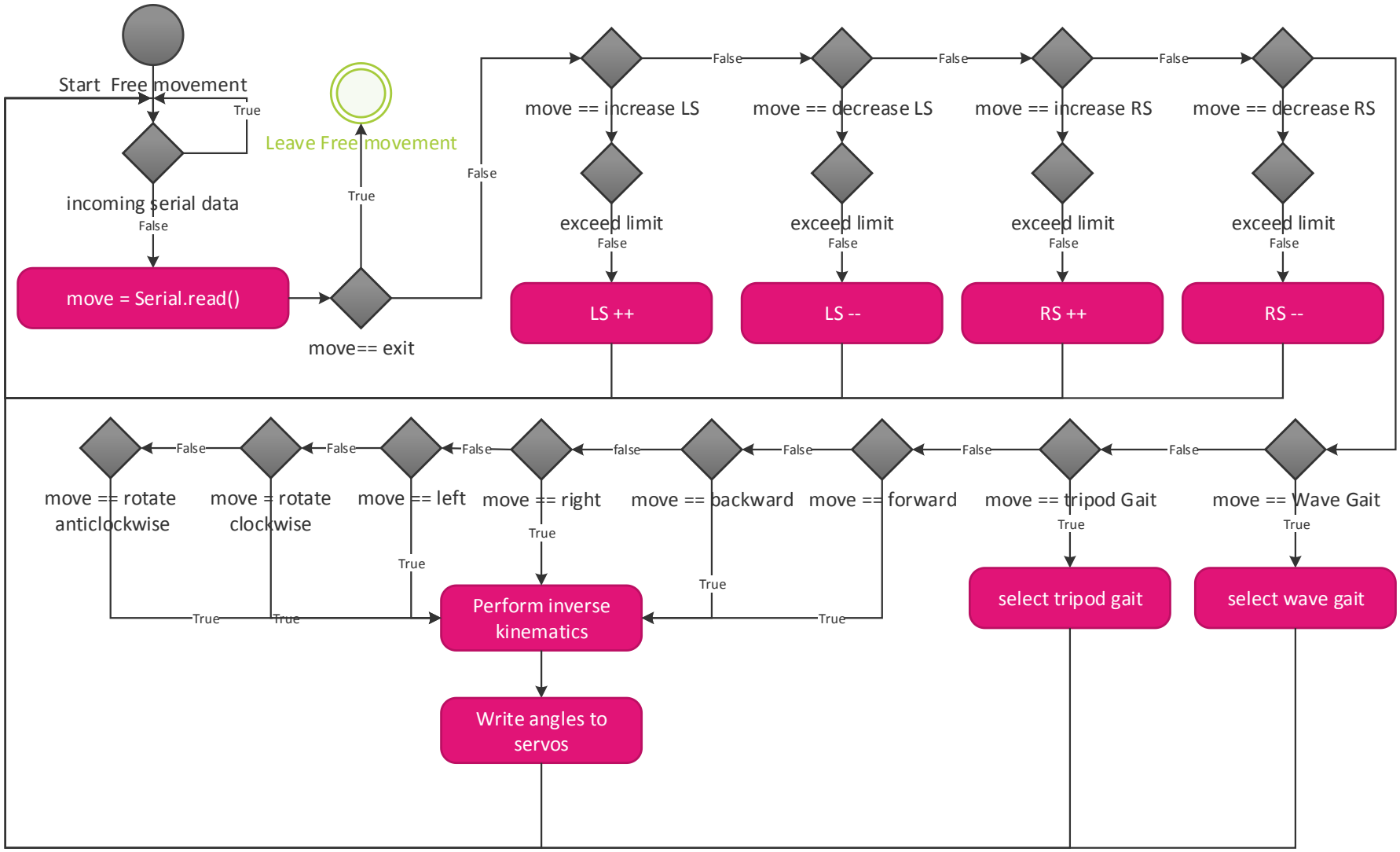

Figure 3.6: Free movement

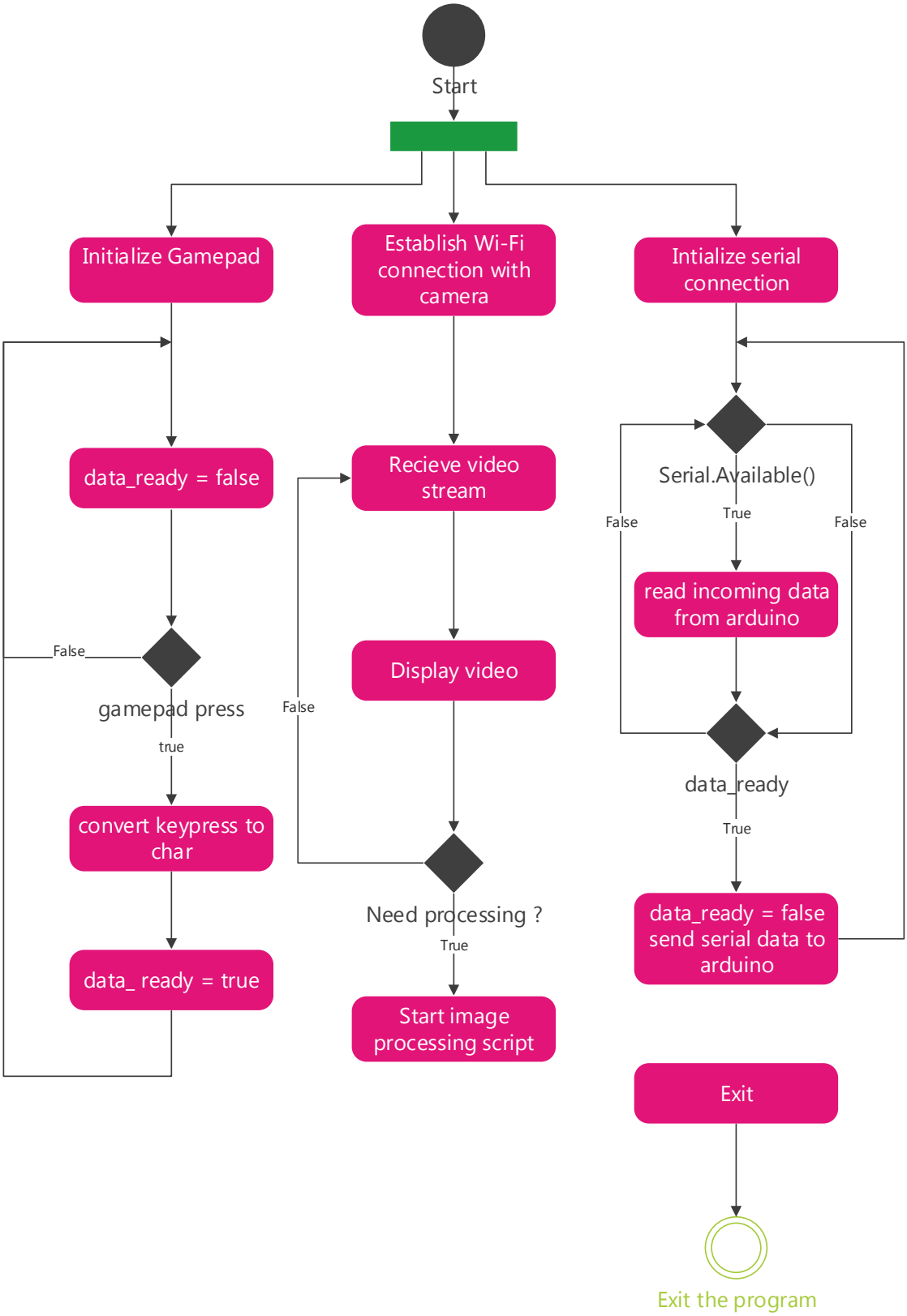

Figure 3.7: Computer Side

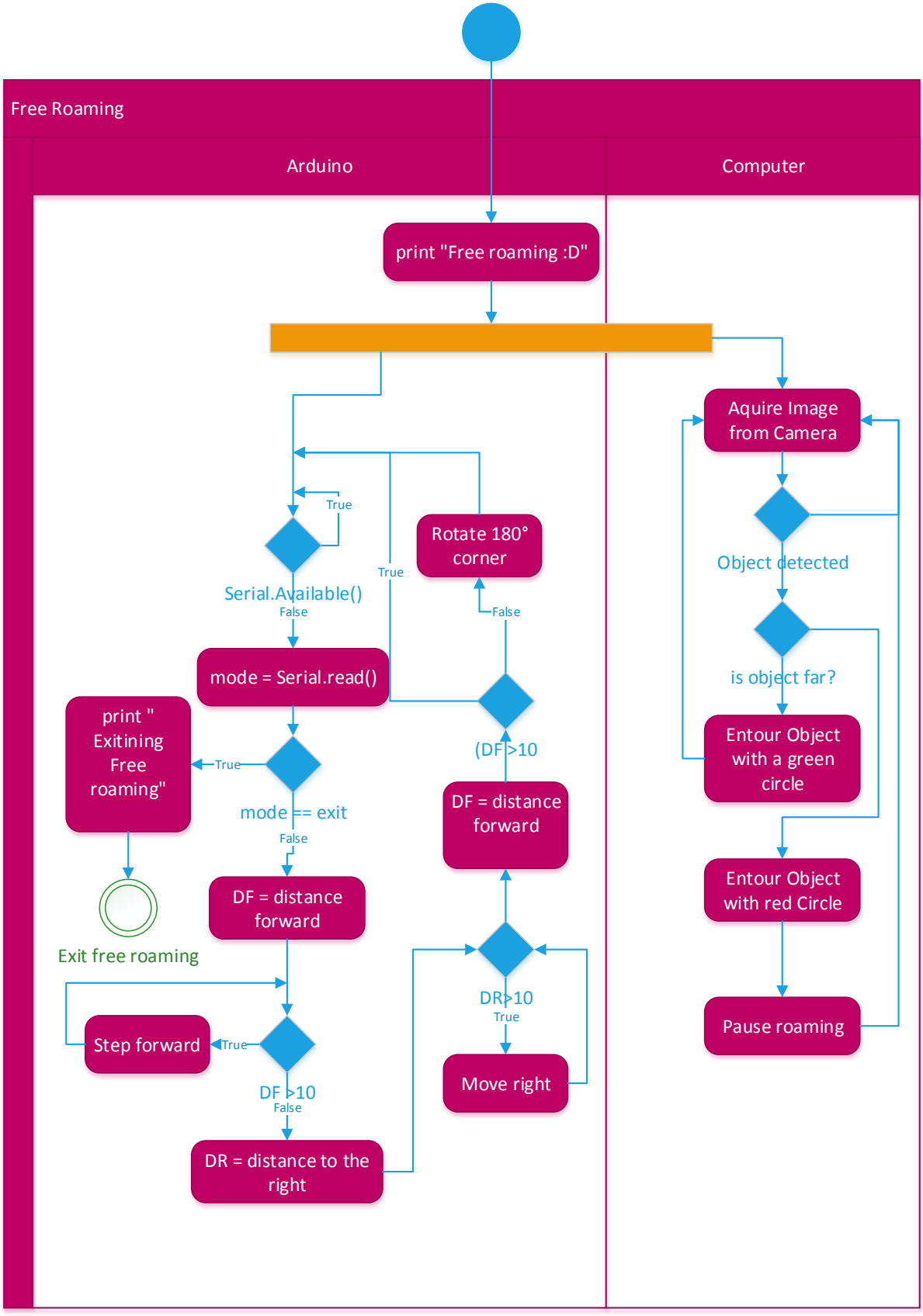

Figure 3.8: Free roaming algorithm

3.4 Image processing

lbIn our humble work we barely scratched the surface of image processing. Our goal was to develop an algorithm capable of detecting and tracking a certain object with a specific shape and colour. For the sake of example, we've chosen a yellow circle as our desired object. The idea is that while the robot is exploring its environment it will keep an eye looking for this object. As an assumption, once the object is detected it will circle it in green and when the robot gets closer to the object, the circle turns red and the robot stops.

To accomplish this we used OpenCV library which stands for *Open source Computer Vision*. It's mainly a software library used for Computer Vision and Machine learning. We've chosen Python as a programming language for this task.

First the video stream is converted into a suitable format, the resolution is also reduced for better performance.

We rely on the colour and shape to detect and track our object. For this reason, we convert the colours from **BGR** *"**B**lue, **G**reen, **R**ed"* to **HSV** *"**H**ue, **S**aturation, **V**alue"*. The reason for this is that BGR colour space defines colours based on their composition of the three fundamental colours, however the HSV colour space defines colours similarly to how the human eye perceive them. After that we split the three components apart and each component is confined into a threshold. The values can be modified to accommodate any colour we want to track. To do this we start with some predefined values, then we keep tweaking till we get the best range for our object. We then *logically AND* the three images to get a binary image where approximately only the object is white and everything else is black. The resulting picture is then smoothed. To identify the circle we use a special function called ***houghCircles.*** Finally the detected circle is drawn on the original image and displayed to the user. The colour depends on how far the object is: green for far and red for near. Activity diagram in figure 3.9 describes this in a better way.

3.4.1Activity Diagram

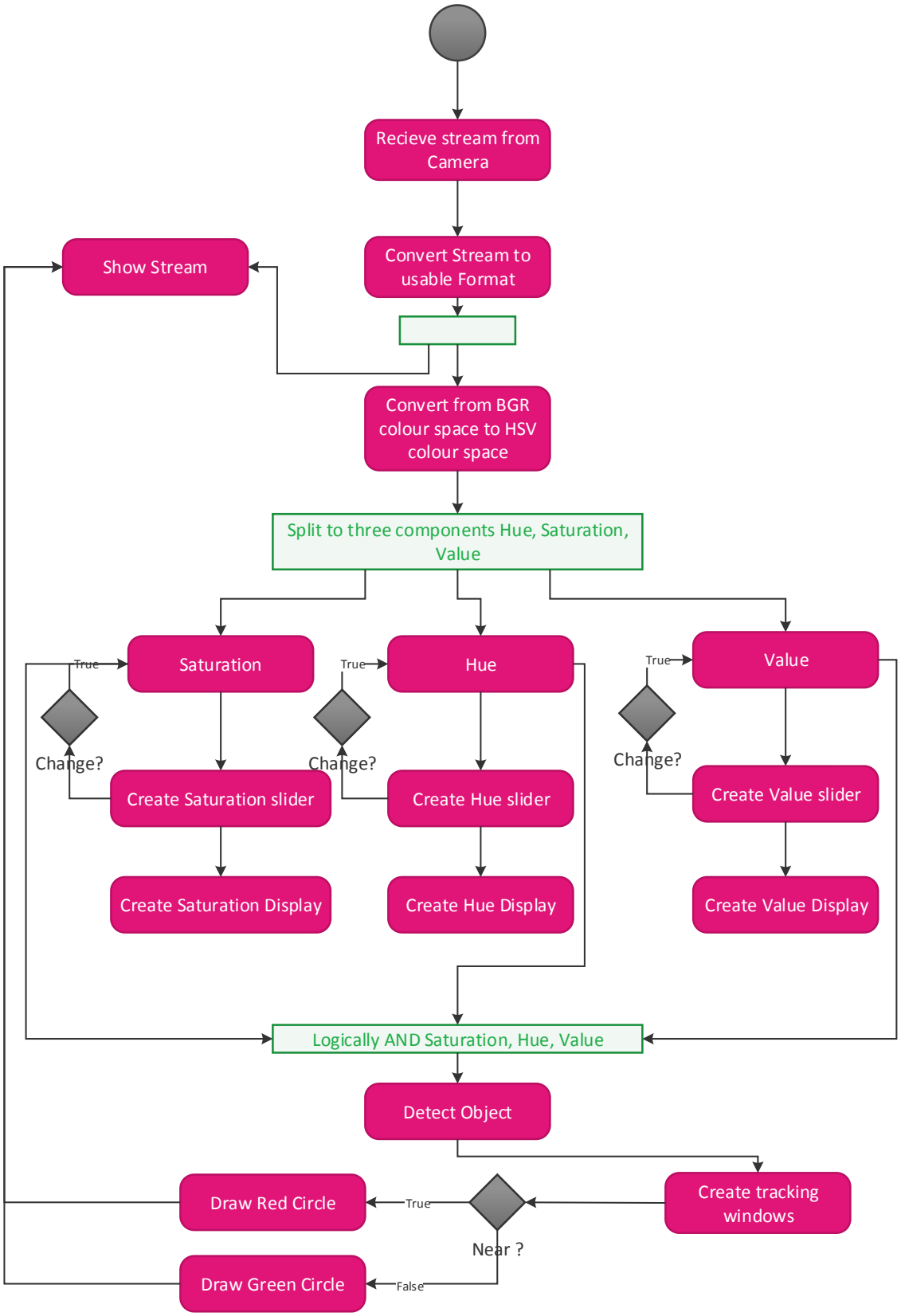

Figure 3.9: Object tracking

Chapter 4 Results and Analysis

To check the robustness of our robot, several tests have been performed. First we connect the different blocks of the system by placing the power supply, plugging the USB cable to the Arduino and the Computer and plugging the gamepad. Then we start the serial link and the software that handles the gamepad presses as seen in figure 4.1.

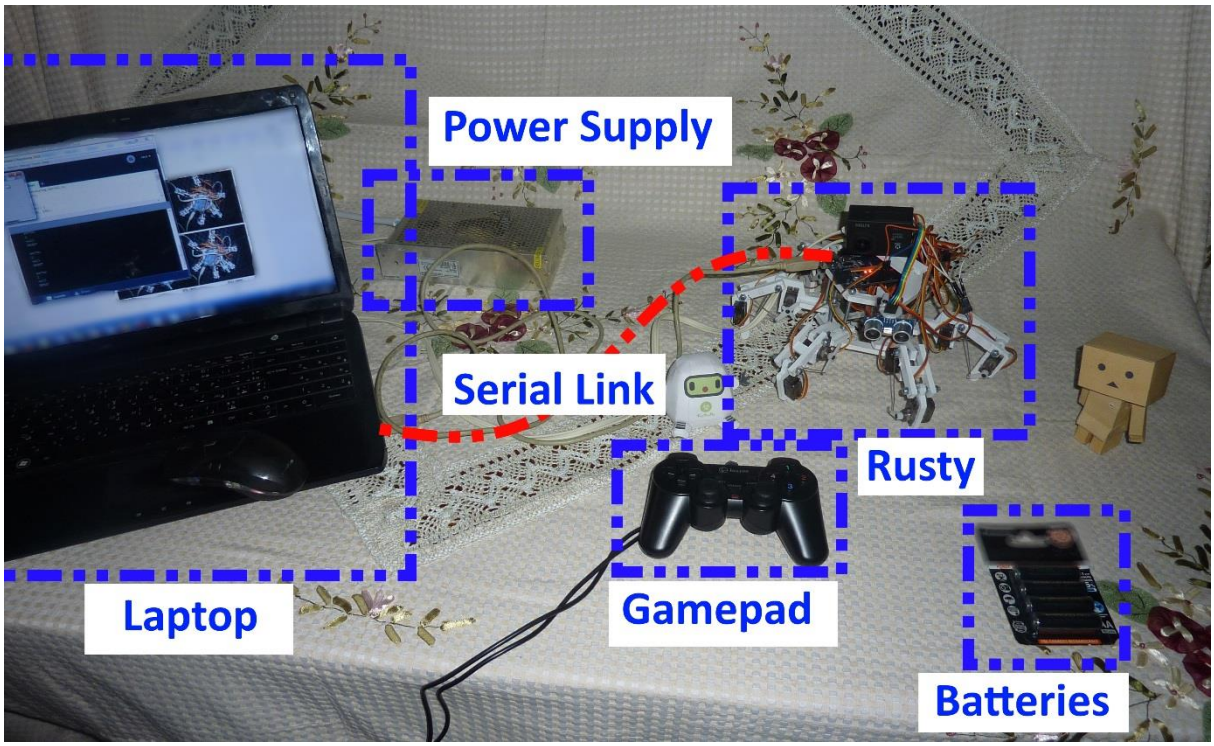

Figure 4.1: System Overview

For this phase we didn't need the camera so we removed it. Once the system is on, the software displayed "***Rusty online* :D"** message meaning that everything is set and working as seen in figure 4.2

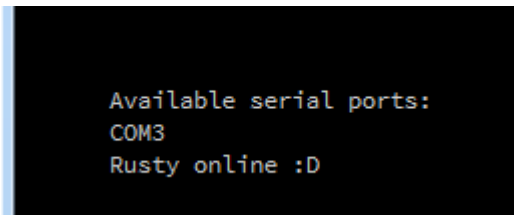

Figure 4.2: Serial link up

4.1 Body Translations

4.1.1 Translation along z

First we started with the body translations. For ***z translation***, we measured the height from the ground to the lowest point of the body. This is useful because it gives us an idea about the height of the obstacles that the robot can walk through without problems. In case we want our robot to go inside some sort of tunnel we simply add to the previous measurements the actual length of the robot's body. Figure 4.3 shows the ***start position***, figure 4.4 shows ***Max z translation*** and figure 4.5 shows ***Min z translation***. Table 4-1 summarizes the findings.

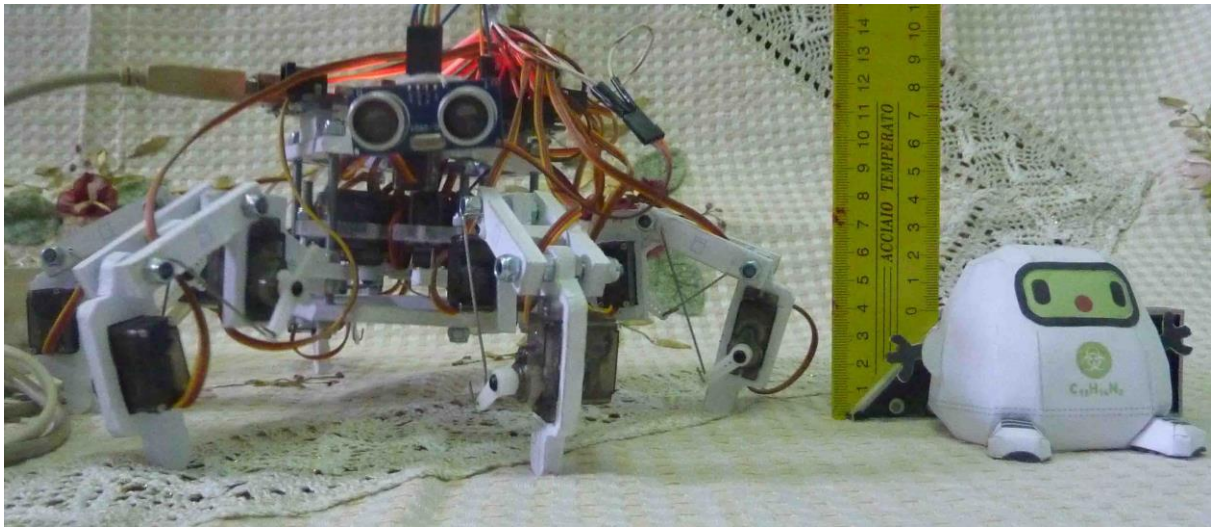

Figure 4.3: Start Position

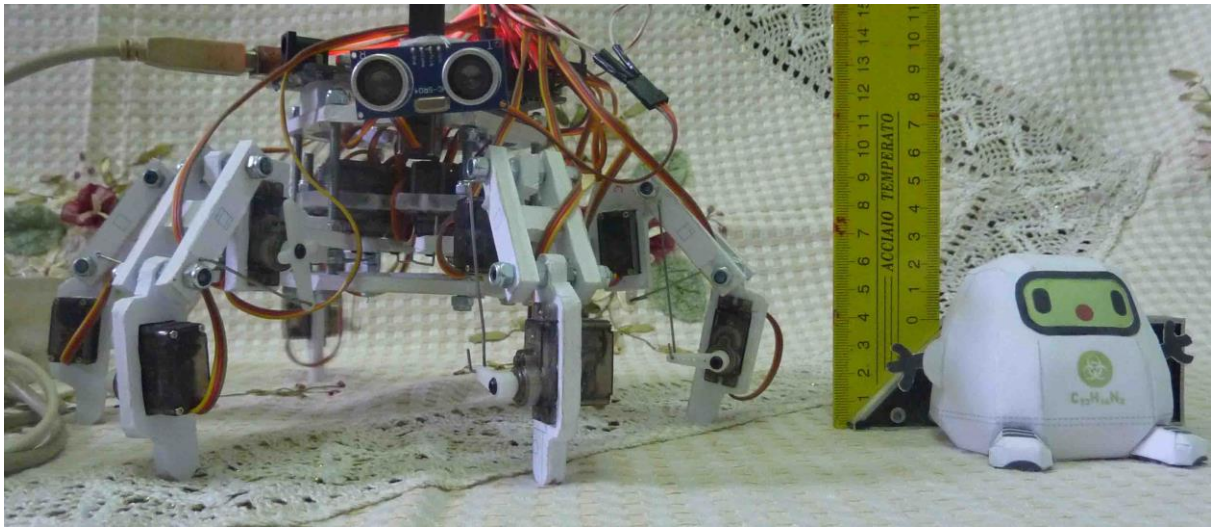

Figure 4.4: Maximum z translation

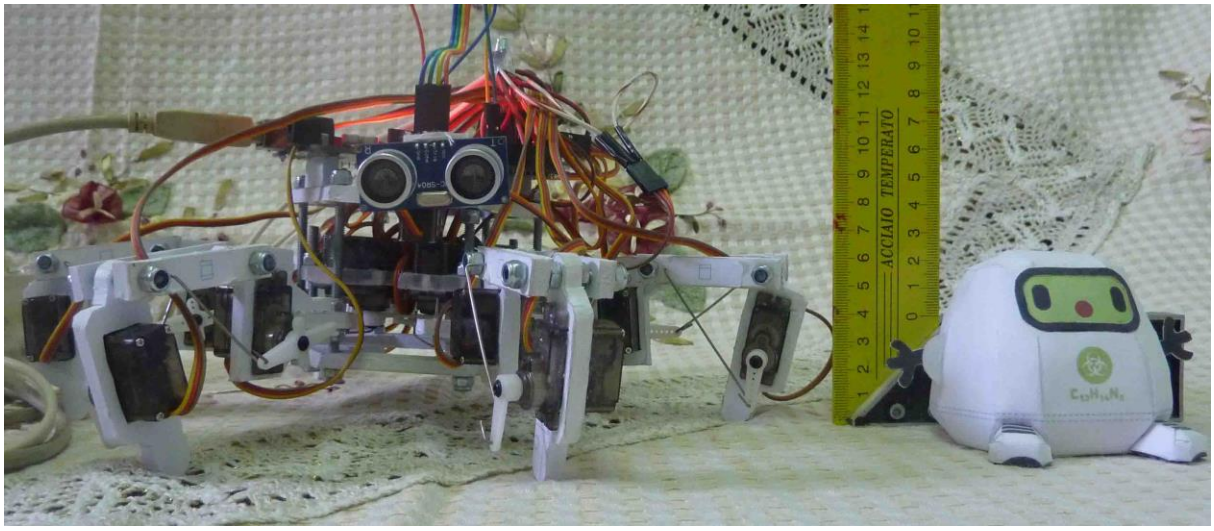

Figure 4.5: Minimum z translation

Table 4-1: z translation

Robot state	***Highest point in the robot (mm)***	***Lowest point in the robot (mm)***
Start position	**120 (180 with camera)**	**35**
Max z translation	**135 (195 with camera)**	**50**
Min z translation	**105 (165 with camera)**	**20**

4.1.2 Translation along x and y axes

Once the translation along z axis has been successfully tested, we tested the body translation along the ***x*** and ***y*** axes. Our reference point is the center of the robot. Figure 4.6 shows the robot in ***Rest position.***

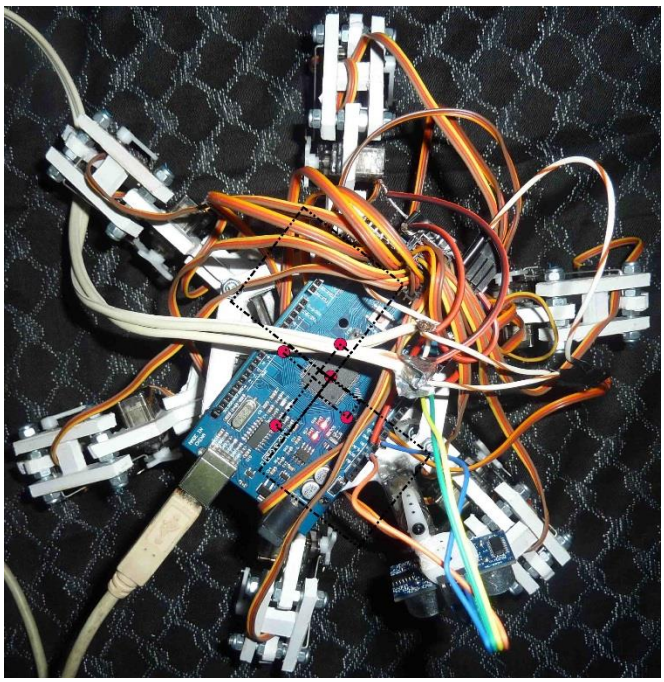

Figure 4.6: Rest Position

Figure 4.7 shows maximum translation along ***y axis***, figure 4.8 shows maximum translation along ***–y axis***, figure 4.9 shows the maximum translation over ***-x axis***, figure 4.10 shows the maximum translation over ***x axis***. Table 4-2 summarizes the measurements.

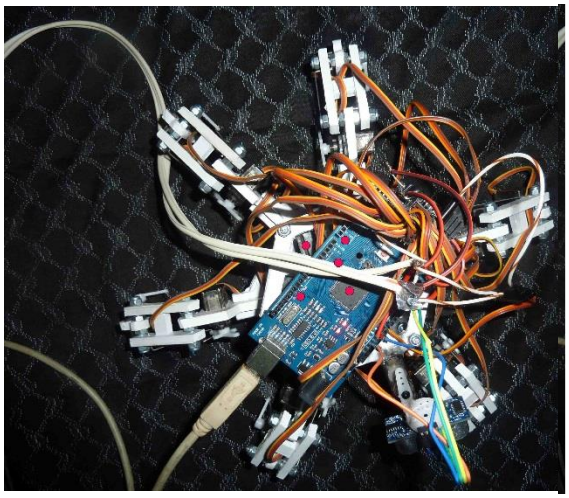

Figure 4.7: Translation along y

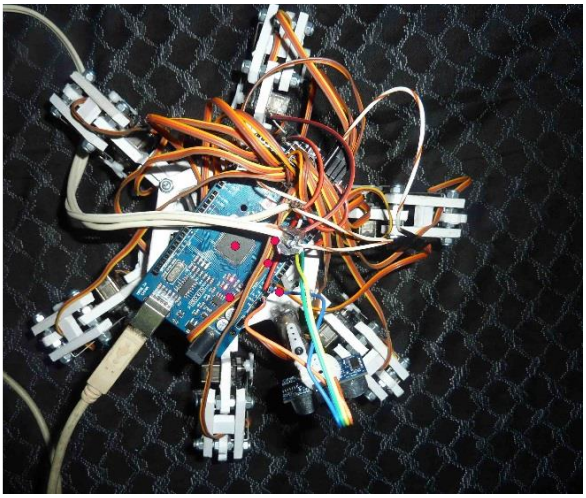

Figure 4.8: Translation along -y

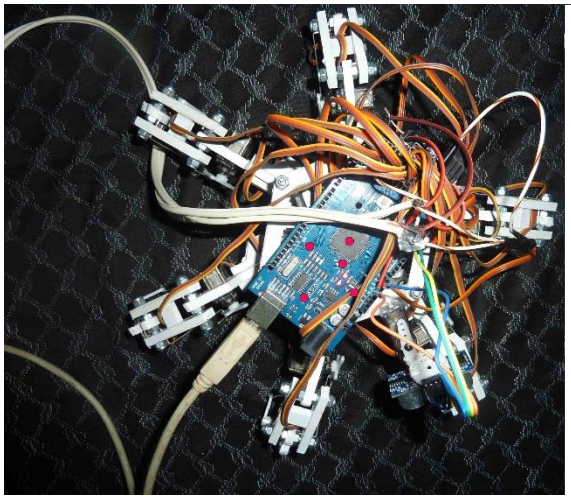

Figure 4.9: translation along - x axis

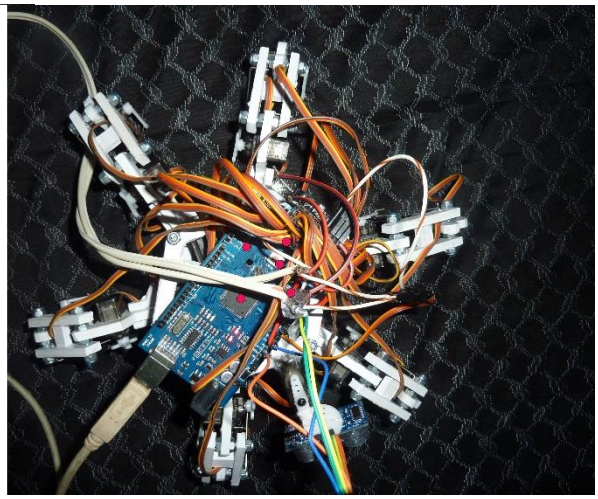

Figure 4.10: translation along x axis

Table 4-2: x,y translations

	Min (mm)	***Max (mm)***
X	**45.5**	**20**
Y	**-25**	**26**

4.2 Gaits

We tested the robot in many terrains. It performed very well on rough terrains. Sand, little rocks, terrain with medium length herbs, carpets and tables, the robot seems to love those terrains. The added friction helps the feet to hold firmly and drag the body along. It had some

problems in slippery surfaces like glass. This is common for all types of robots and new algorithms are currently developed to detect slippery terrains and avoid them if possible.

4.2.1 Speed

4.2.1.1 Linear speed

The robot has ten levels of speed. We tried the tripod and wave gaits for each speed level and calculated the actual speed of the robot at each level.

Table summarizes the results and figure 4.11 displays them as 2-axes graph to see the relationship between the variation of the speed as the level decreases. On the software side the speed levels are related linearly. The experimental results confirm this with a deviation of less than 15% from a perfect straight line.

Table 4-3: Speed levels

	Speed cm/s									
Level	10	9	8	7	6	5	4	3	2	1
Tripod	10.66	10.12	9.85	8.14	7.80	7.37	5.90	5.47	5.10	4.63
Wave	3	2.2	1.8	1.7	1.2	0.95	0.7	0.6	0.5	0.867

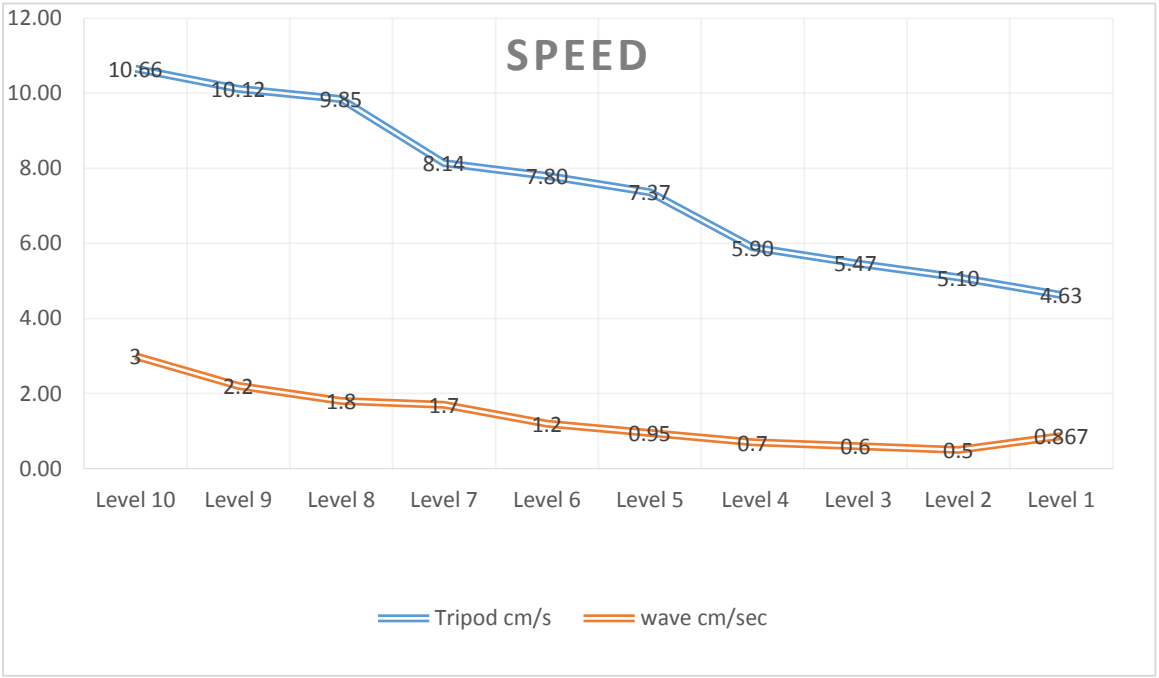

Figure 4.11: Speed levels

4.2.1.2 Rotational speed

Our robot can walk in any direction. This feature reduces the need for rotation. However there are times when we need the robot to rotate as in the case when the mounted camera is used.

Our robot is capable of rotating in many speed levels. The following table summarizes our experimental results and figure 4.12 displays its corresponding graph.

Table 4-4: Rotational speed

	Angular velocity deg/sec									
Level	10	9	8	7	6	5	4	3	2	1
Speed	12.97	11.84	7.11	5.86	4.65	4.05	3.78	3.23	3.05	2.63

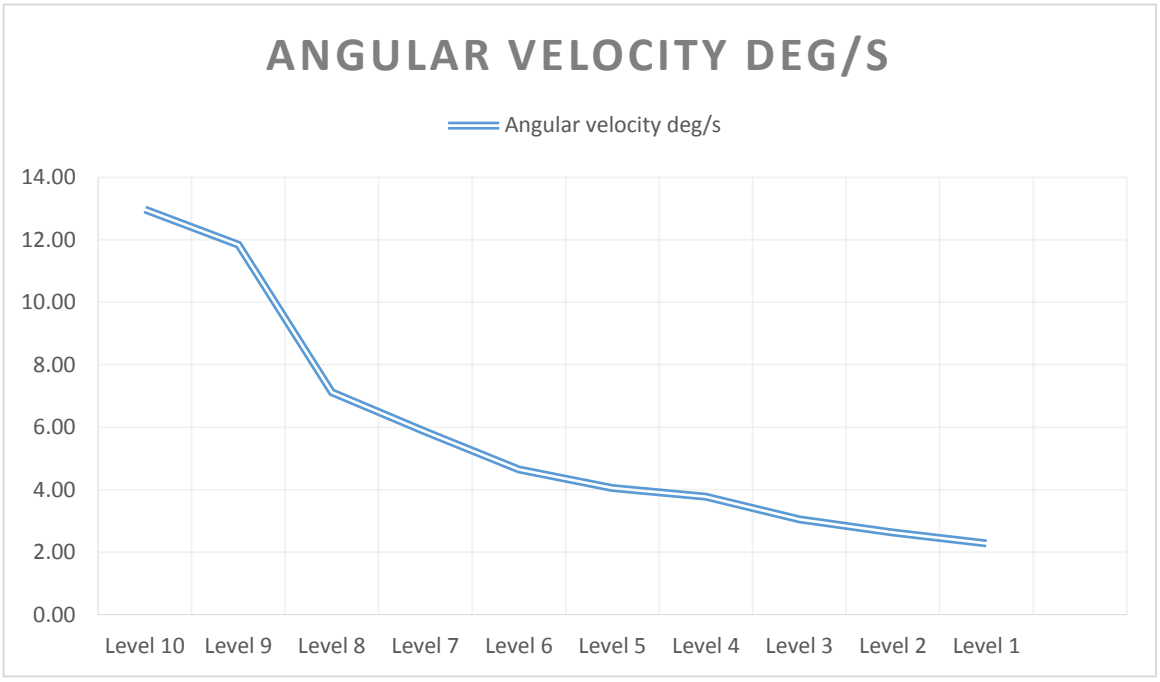

Figure 4.12: Rotational speed (deg/s)

4.3 Free roaming algorithm tests

4.3.1 without Camera

We tried the algorithm without a camera and we found that it successfully avoided obstacles. It also avoided getting stuck in corners. Figure 4.13 shows the serial output while the robot was roaming.

```
Available serial ports:
COM7
Rusty online :D

ports:
1
null
Free roaming :D

ports:
1
Sonar looking forward

Distance to obstacle: 20

Done with one step

Distance to obstacle: 17
```

```
Done with one step

Distance to obstacle: 14

Done with one step

Sonar looking right

Corner detected rotate to avoid

Done with one step

Sonar looking forward

Distance to obstacle: more than 5 meters

Done with one step
```

Figure 4.13: Run example without Camera

Refer to the previous figure, when the serial connection between the robot and the computer is established, we chose the free roaming mode. In this mode, the robot starts with the ultrasonic sensor looking forward. It detects an obstacle and measures the distance to the obstacle. With each step, the distance gets smaller which means that we're advancing towards the obstacle. Once the obstacle is 10 cm far the ultrasonic sensor turns right to check if there's no obstacles. If there isn't any it will walk to the right. On the experiment the robot encountered a corner so it rotated 180 degrees and continued roaming.

4.3.2 with camera

This time we placed the camera. First we adjust the thresholds of the Saturation figure 4.14, Hue figure 4.15 and Value figure 4.16 to match our object. If that's done properly we'll see

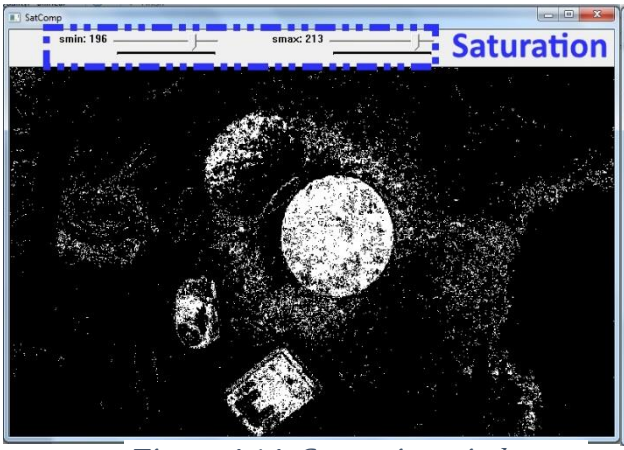

Figure 4.14: Saturation window

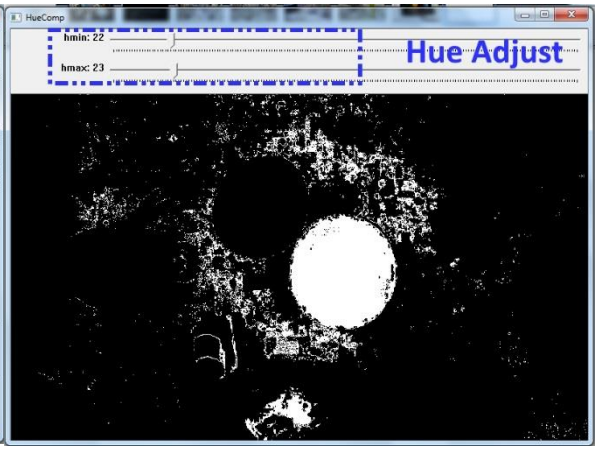

Figure 4.15: Hue windows

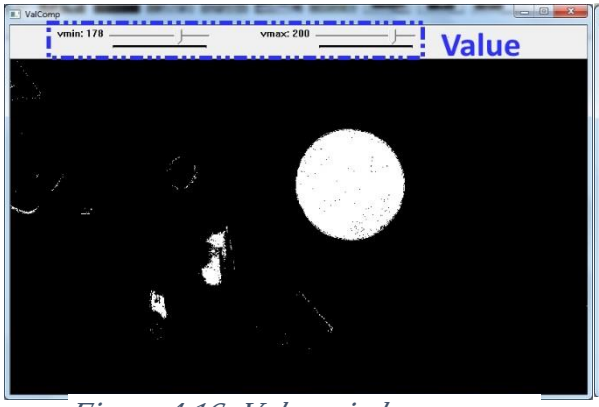

Figure 4.16: Value window

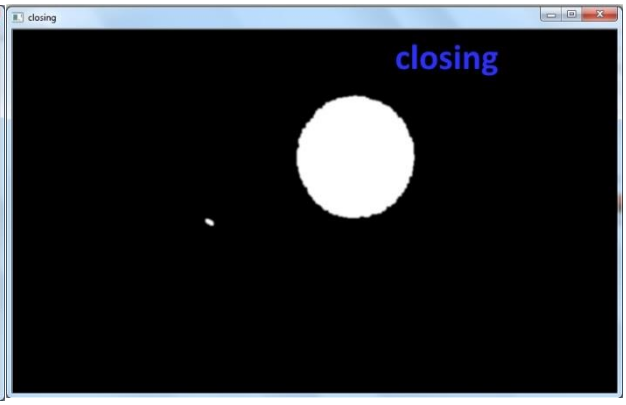

Figure 4.17: Closing window

only our object in the closing window as seen on figure 4.17. The colour of the circle depends on the distance. Figure 4.18 shows a red circle when the object is near and figure 4.19 shows a green circle when the object is far.

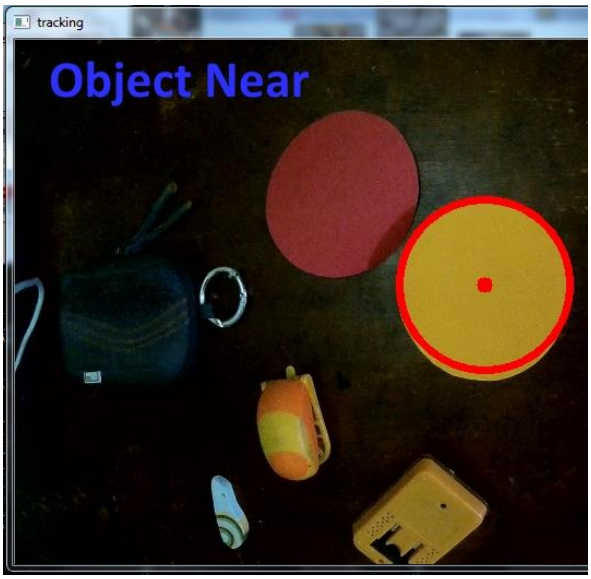

Figure 4.18: Near Object

Figure 4.19: Far Object

Figure 4.20 shows the serial output when the robot detects the object. It then waits for the user to either take the object and the robot continues roaming looking for other objects or exit the free roaming mode. In our example the user decided to exit after the robot has found the object.

```
ports:
3
null
Found object :D

ports:
2
null
Exiting free roaming
```

Figure 4.20: Object found

Conclusion

This work was about the design and implementation of a six legged (hexapod) autonomous robot capable of obstacle avoidance using a single ultrasonic sensor and object tracking using a mounted camera.

We built two frames for the robot: one made from *6mm Plexiglas* and the second from *5mm Forex*. The first design is stronger with the drawback of the added weight and the time consuming manual work to cut and refine the parts. The second frame is lighter and easier to cut and refine with the disadvantage of the reduced resistance. However, unless if someone wants to compete in Robot Combat League, both frames are strong enough for all the intended uses.

The robot has two modes of use. The first mode is user controlled and the second is full automated one. The user controls the robot using a gamepad or the computer's keyboard. The user can also get a live feed from the camera at any time. The camera can stream over a range of 100 meters. It can stream to computers and smart phones.

We developed a program that runs on the robot which relies on an Atmega 2560 microcontroller embedded in an Arduino mega board. The main part is an inverse kinematics engine capable of generating all sort of movements. For instance it can generate the tripod and wave gaits, it can also generate body translations. Using the ultrasonic sensor the robot can detect and avoid obstacles without getting stuck. On the computer side we have several programs working independently. One is responsible for translating the gamepad presses. Another is responsible for the serial communication with the Arduino. The third one is a python script that manages everything related to the camera. For instance it detects an object from its shape and size and tracks it in real time.

We performed many tests on different terrains. The robot performed well on rough terrains by adjusting the speed of the gait accordingly.

This work serves as a prototype that can be easily adjusted to fit the intended use. This robot can be used in rescue missions to find survivors. It can be used to discover and map places that aren't accessible to human beings like caves.

Future perspectives

For start it's advisable to add the components that we couldn't due to their inaccessibility to us. A magnetometer would give the robot a sense of direction, a good wireless transmitter would

eliminate the need for the USB cable and a GPS would add a whole new domain of applications. We could also replace the Arduino with something more powerful in order to do inline image processing. Obstacle avoidance can be done using the camera and the ultrasonic sensor would only care about the unexpected obstacles. We can also experiment with adaptive gaits and flexible muscle-based locomotion

References

[1] May, R. M. How many species are there on earth? *Science*, Volume 241: 441-1449. 1988.

[2] Csonka, P. J. and Waldron, K. J.Technology Developments: the Role of Mechanism and Machine Science and IFToMM.Ceccarelli, M. (ed.).Chapter A Brief History of Legged Robotics. Springer Netherlands, 2011, pp. 59-73

[3] MIT Leg Laboratory. "3D One-Leg Hopper (1983-1984)." *3D One-Leg Hopper*. N.p., n.d. Web. 21 May 2016. http://www.ai.mit.edu/projects/leglab/robots/3D_hopper/3D_hopper.html

[4] Morazzani, I., Lahr, D., Hong, D.W., Ren, P., "Novel Tripedal Mobile Robot and Considerations for Gait Planning Strategies Based on Kinematics," *Recent Progress in Robitics: Viable Robotic Service to Human*, pp.35-48, Springer-Verlag Berlin Heidelberg, 2008

[5] Boston Dynamics. "Boston Dynamics: Dedicated to the Science and Art of How Things Move." *Boston Dynamics: Dedicated to the Science and Art of How Things Move.* N.p., n.d. Web. 21 May 2016. http://www.bostondynamics.com/robot_cheetah.html

[6] Besari, A. R. A., Zamri, R., Prabuwono, A. S. and Kuswadi, S.*Intelligent Robotics and Applications: Second International Conference, ICIRA 2009, Singapore, December 16-18, 2009. Proceedings.*Xie, M., Xiong, Y., Xiong, C., Liu, H. & Hu, Z. *(ed.).*Chapter The Study on Optimal Gait for Five-Legged Robot with Reinforcement Learning.Springer Berlin Heidelberg, 2009, pp. 1170-1175

[7] Waldron K. The Adaptive Suspension Vehicle / Kenneth J. Waldron, Robert B. McGhee // IEEE Xplore – Control Systems Magazine, IEEE. – 1986. – N 6. – P. 7-12.

[8] Ben Greer. "Lab Notebook." *: [HEX] Inverse Kinematics*. N.p., n.d. Web. 21 May 2016.

[9] Siegwart, Roland, Illah Reza. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. Cambridge, MA: MIT, 2011. 20-21. Print.

[10] Albarral, J. L. "Implementation of a Driver Level with Odometry for the LAUTON III Hexapod Robot." *Climbing and Walking Robots: Proceedings of the 7th International Conference CLAWAR 2004*. By E. Celayade. Berlin: Springer, 2005. N. pag. Print.