Development of hexapod robotic systems

CS39440 Major Project Report

Author: Oskar Lipienski (osl1@aber.ac.uk)

Supervisor: Dr. Dimitris Tsakiris (dit5@aber.ac.uk)

10th May 2020

Version 1.0 (Final)

This report is submitted as partial fulfillment of a BSc degree in Artificial Intelligence and Robotics (GH76)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

Name: Oskar Lipienski

Date: 10/05/2020

Consent to share this work

Name: Oskar Lipienski

Date ………………………………………………

Abstract

People have always been looking at nature with envy. We wanted to run faster like a cheetah, swim like a fish, and fly like a bird. This is why we take so much inspiration from nature when creating new technologies, learning, and building from biological creatures - nature's greatest invention. This way we were able to fly, swim, and travel faster than we could ever imagine. But with that achieved we do not stop inventing and progressing, while still looking at animals as our best inspiration.

Robotics is one of the most important branches of technology when it comes to replacing humans when doing tasks in hazardous environments, like space or areas of radiation. Robots are more precise than people, they do not need to rest, can be controlled over great distances, or just do tedious and repetitive tasks.

This project is marrying those two things to create a robot inspired by 6-legged creatures, to see how design and control strategies that we can observe in nature can be used and how do they compare and apply to those created by humans. It also aims to develop a hexapod robot that would be able to autonomously explore the environment and find its way around it.

Contents

1. Background, Analysis & Process

This project includes the creation of particular parts of the robot in CAD software, the design of the whole robot in a simulation environment with motors and sensors, and writing the code controlling the robot. To approach a project consisting of so many various tasks the background knowledge has to be vast, reading extensive and engineering skills on point. The scope of the work implies that even the smallest mistakes will blend into the next stages of work and influence the final outcome.

1.1. Background

The first thing to do when approaching a project like that is getting some background knowledge, information that will enable us to identify the work that needs to be done in order to start the work. It is also important to do thorough research in the area of tools that will be used as changing them in the middle of the project is often more expensive in time and resources than changing basic concepts of the project.

To program a robot, the first step is to design a robot. Thus the vast majority of the background reading has been spent on discovering different approaches to designing a robotic system. With some concepts established, it was time to create a CAD model of parts of a robot, which then can be exported into the simulation. Before doing any work the area of CAD creation software was explored. It wasn't very crucial as it was only a tool to create some shapes that can be later assembled in the simulation. As I have done mechatronics in high school I was very happy to work with CAD software again as I know it along with hydraulics, pneumatics, electronics, and electrics very well.

After the model is done it was important to research different simulation tools and middleware for the robot. It was crucial to pick one that is efficient, easy to use, and compatible with middleware of my choice. By the time of the choice of the simulation tool, it was almost certain that ROS will be used as a middleware so one of the most important requirements was compatibility with it and Ubuntu 18.04 because that is the version of Linux on which ROS runs best. After choosing a simulation tool and deciding ROS is the middleware to go with those two can be set up to run together on one system.

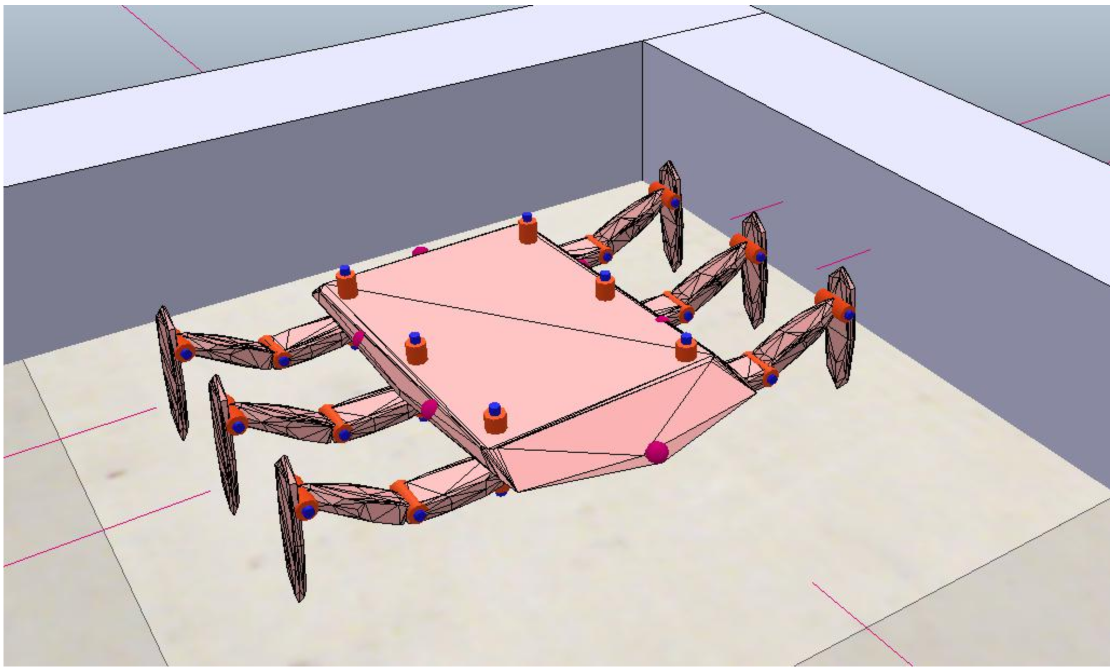
The next choice, which is the choice of the program development tool, is the least important one as it will only impact the comfort of the programmer creating the program.

Then the time came for a research in the area of movement control of the robot. The difference between a wheeled robot and a hexapod is, among others, the number of motors the program has to actuate to perform a certain action. If we want to move robot a meter ahead at fixed velocity all we have to do is send a command to a controller that will power at least one motor and move the whole thing. However, the fact that our robot has six legs and in each leg, we have 3 motors for actuation implies a completely different approach. To move the robot forward we have to actuate 18 independent motors in a synchronized fashion so the robot will not fall over or trip over an obstacle. This can introduce a set of errors all of which have to be dealt with.

Finally, the last step was to use behaviors like walking and turning and to use them to navigate in an environment. This is the most advanced step of the work, which might seem easy, but because of the complexity of the motion of the robot, it is extremely difficult to navigate and locate a six-legged robot.

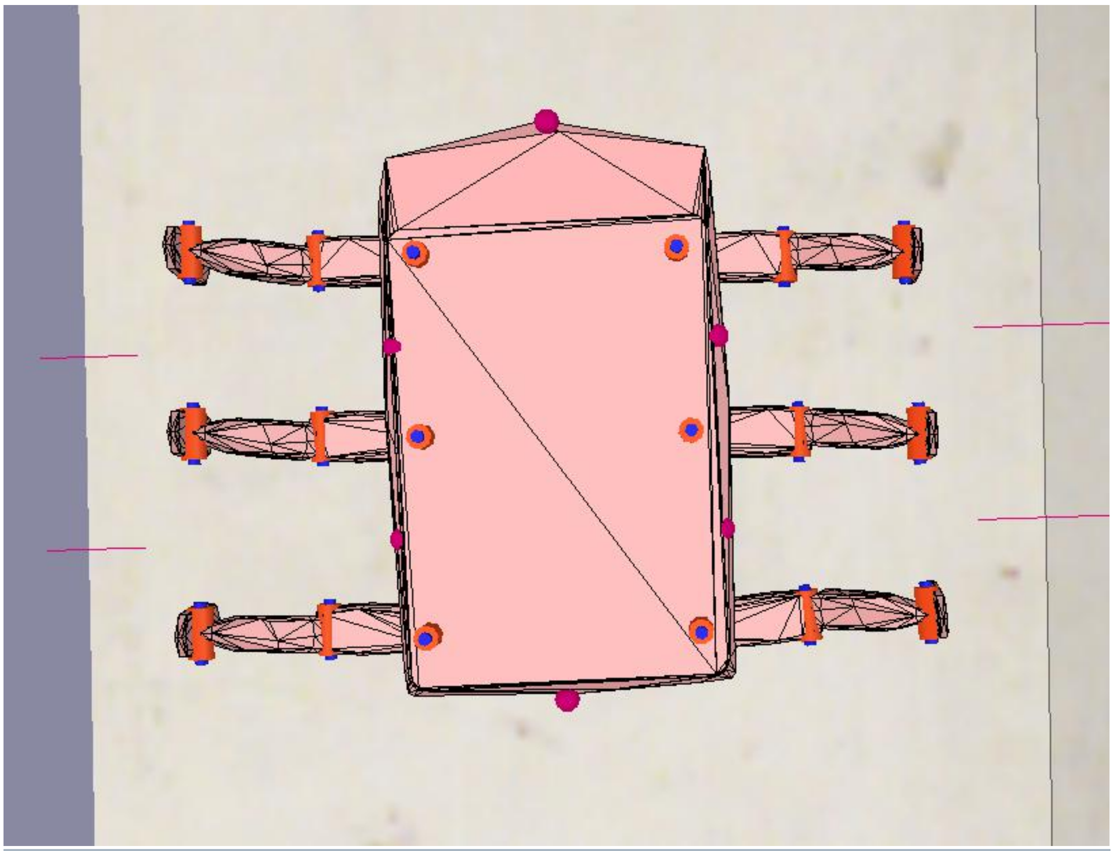
1.2. Analysis

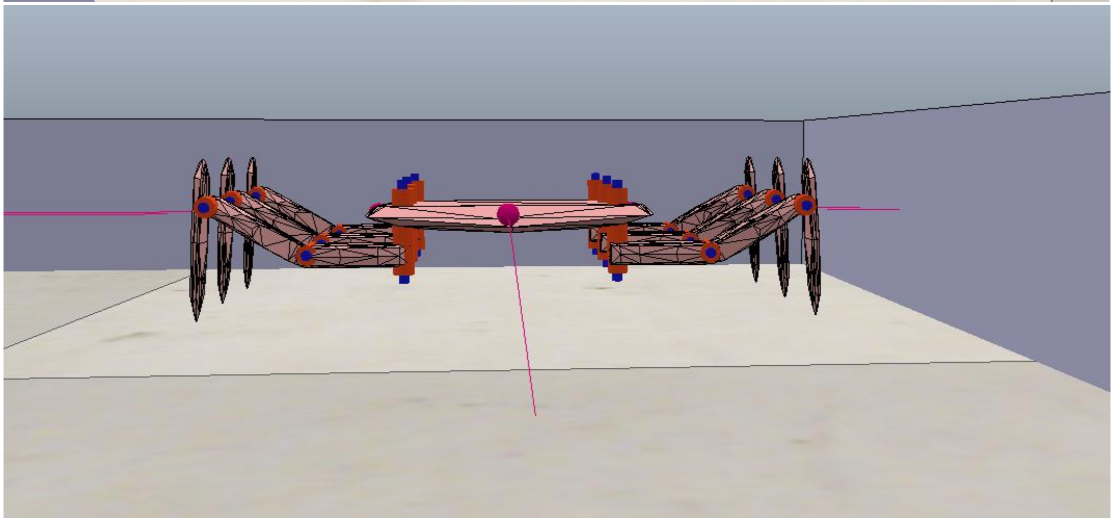
The first objective was the design of a six-legged robot that will be able to perform tasks like moving around, avoiding obstacles, and localizing itself in the environment. This involves using electric motors and sensors. Six legs will provide robot enough stability even with two legs not touching the ground and sensors will let it see the environment and plot its own position in space.



After researching the area of spider-like robots it became clear that 3 joints per leg are sufficient value. The first two degrees of freedom will enable the robot to lift its leg and move it to the front and back while the third degree will help it reach for the objects that are higher or further away. The third motor is also helping with keeping balance as it ensures that the tip of the leg is touching the ground at the right angle.

The next important decision to make is the length of each leg and the body of the robot. In the case of this project, the length of legs is not as important as the proportions of length of legs and length of the body. If legs are too short the robot will drag its bottom on the ground while making very tiny steps. If legs are too long in comparison to the body the area of movement for each leg will be very small due to clashes with other legs. Taking that into account it has been decided to go with the dimensions shown on the graphic below.





Sensors are the eyes, skin, and nose of the robot. If we pick wrong we might end up in a situation when we can not perform a certain action because we lack proper data about our environment. That is why it is so important to choose the correct ones. The robot designed for the purpose of this project at first had only two front sensors returning False value when there was nothing on their way and True if there was something. Although, as soon as functions like localization were about to be developed it became clear that 2 Boolean front sensors are way too little for its needs. That is why the new version has 4 IR proximity sensors. One for each side of the body. Each sensor is scanning to look for obstacles and place them on a map drawn by the robot as it is exploring the environment.

After the robot is designed programming has to be done. The first thing to do is the implementation of the simplest behaviors like walking. As mentioned above even implementation of a trivial action carries a certain amount of complexity as to perform it 18 different motors have to be actuated.

There are different types of gaits for hexapod systems. A gait is a pattern in which the robot will move its legs in order to perform an action. Most popular are the tripod gait, where a robot is using 3 legs at a time to perform a step while the other 3 legs are touching the ground providing stability. Another type of gait is the gait where 5 legs stay in touch with the ground while one leg is performing a step. The rest of the different gaits can be described as variations of the two. For example, instead of lifting only one leg, we can lift another leg in the middle of the movement of the first one. This makes the movement of the more natural and fluid. For the project, I decided to go with tripod gait for movement in a straight line and 5-point gait for turning as it is way more accurate than tripod gait.

The last step of the project was improving upon a controller to achieve autonomous exploration and localization in a created environment. To do this a number of different strategies were analyzed. Finally, after some time trial and error robot is moving around in an environment in a fashion so the robot first checks if it can go towards the X coordinate if there's an obstacle on the way it will try and go towards Y coordinate. It will do it until it reaches the goal.

1.3. Process

Not many process methodologies will be suitable for this kind of project. It consists of a few phases:

- Design of the robot
- Creating a robot inside a simulation
- Writing controller for basic behavior
- Improving on a controller to achieve autonomous exploring and localization

The scope, in this case, is big enough that each of the phases has been treated as an independent process. This way it makes it easier for a person to manage all the work.

In the case of this project, each phase has been treated like it was a process managed with the waterfall model. Each phase was carefully planned, analyzed, and finally implemented as a stand-alone project. This methodology allows for working on a few different things at a time, which is especially important when doing big, long projects that might get boring. Developing two phases in parallel has also one more advantage, it makes it easier to eliminate things that would be a problem if the phases would be developed one at a time. Unfortunately, there are also downsides. Not every phase can be developed in parallel. For example, the controller can not be improved if there is no controller.

Therefore, by looking at the whole project, it has been developed using agile development as all the phases had to be synchronized together in one cohesive system even after finishing work on a certain phase due to errors that have been encountered during later work it was very often a must to go back to a finished phase and improve upon it.

2. Design

The project consists of four independent aspects and each of them has to be described in detail. The first being the design of the robot itself. It includes the placement of sensors, motors, and the way they are connected together. The second one is the process of building a robot inside the simulation. As the reality of the simulation is very often different than what has been planned some adjustments had to be applied. The next aspect is the design of simple behaviors like walking and turning. This was especially complicated because of the design of the robot. 6-legged systems are particularly complex because all legs have to work in sync for a robot to not fall over and be in balance all the time. The last thing to consider is the design of algorithms that will enable the robot to autonomously walk towards the target that has been set in the 2-dimensional operating space of a robot.

2.1. Design of the robot

The design of the robot has started with research in the area of the hexapod robots. After some investigation, it became clear that 6 legs and 3 servos per leg are the best solution. With 3 motors it was easy to control the height on which the robot will operate without changing the angles of the part that will touch the ground, which is important because the more repetitive are the moves of the robot the more precise the moves will be, which in later stages of the project made a lot of things easier. A good example would be the algorithm that is localizing the robot. Less certainty in terms of where the robot is after taking a few steps can translate to much more work to and sensors that can help establish a robot's position.

The next step was the CAD project. Each part has been modeled by hand in the AutoCAD 2020 software. It was crucial that each part is the right size and then converted to triangular mesh. Converting parts to triangular meshes was the most troublesome operation in the whole process but it was a requirement of the simulation software. Later in the project, it was needed to go back to CAD designs because of the sizes of different parts. During the design phase the length of legs with respect to the body was too long, so when the model was moving the legs were crashing into each other even when making very small steps, so it was obvious that the body needs to be longer to make space for legs to move around.

The placement of the sensors was specified by trial and error. First, there were only two front sensors making sure that the robot will not bump into any obstacle. After some time it became clear that more than two sensors with Boolean values are needed. This is why they have been replaced with 4 proximity sensors measuring distance in every direction. Now they are being utilized to properly locate obstacles on the internal map. There is also a set of sensors that are helping the robot tell in which position each motor is. Each servo has an encoder that helps to tell in what position is each motor and through that knowledge, the robot is able to synchronize all the servos together.

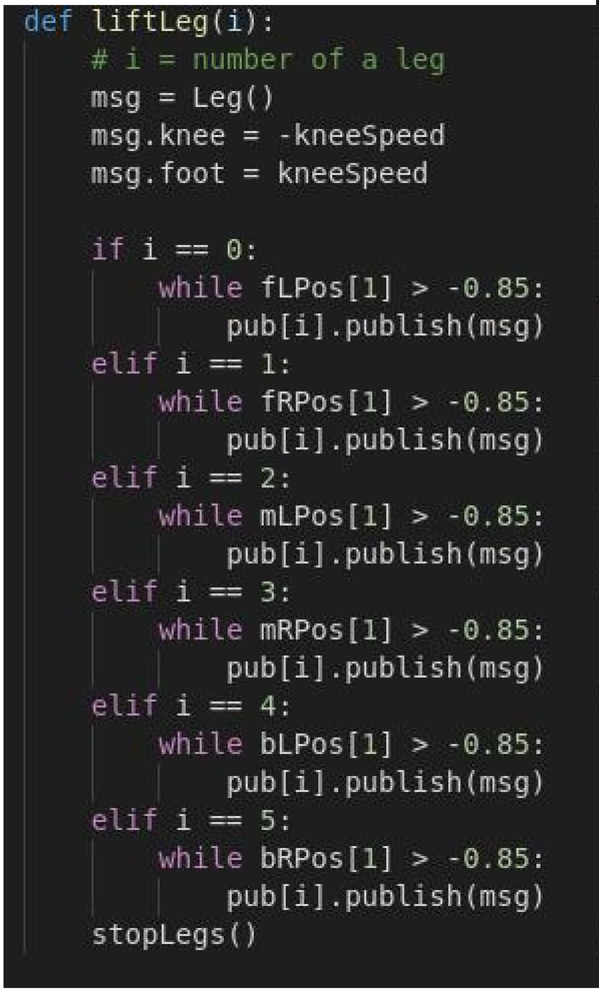
2.2. Design of the simulation

It has been decided to write a different section about the simulation of a robot because primarily the robot has been built inside of it. Only parts like foot or body were modeled in CAD software. Servos and sensors have been implemented and configured already inside the simulation software.

Each part has been manually put in place and scripted already inside the CoppeliaSim Edu V4.0.0 program which is the simulation software of choice. This way it was way easier to put together a well build robot. The opposite of this approach is writing a URDF file which is a textual description of a system. CoppeliaSim is advanced enough to set all the details about the robot already in the program. Details like traction, material from which the part is built, the weight of every part can be determined within two clicks and that is the reason why this particular software was so appealing to use.

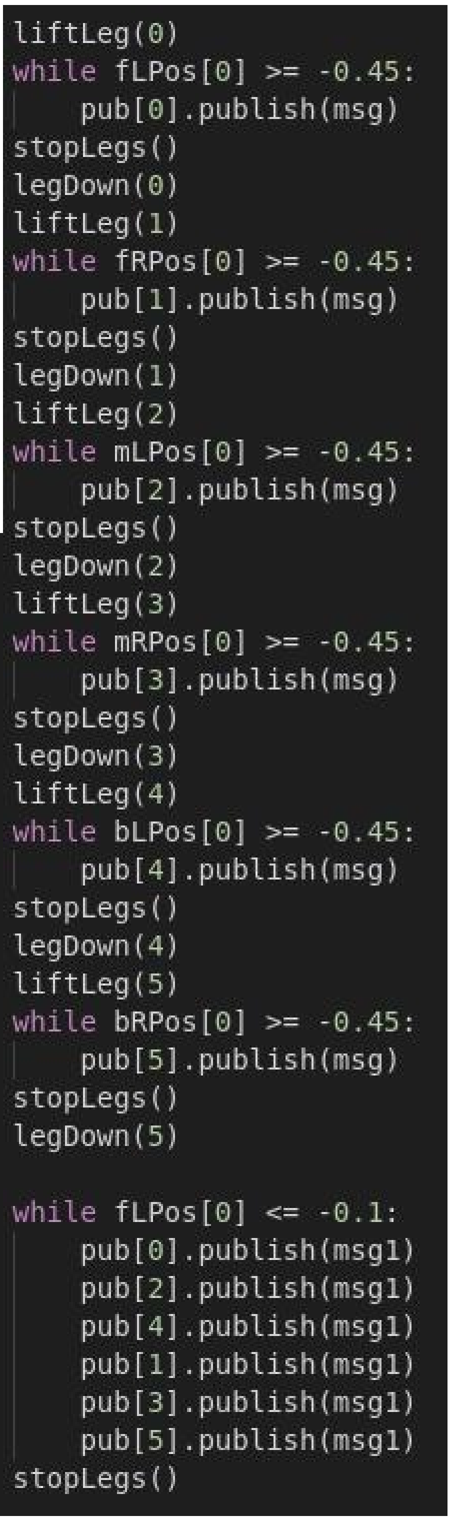
After the robot has been put together in a program it was necessary to write a script that would connect the simulation with ROS. The language in which the script has been written is called LUA. All the publishers and subscribers that appear in ROS after the simulation has been started are scripted inside CoppeliaSim and handled by ROS interface plugin. However, as it would be difficult to publish all the motors positions and velocities as separate values for each motor, for a custom hexapod robot it was necessary to recompile plugin to handle custom messages that have been created to use multiple servos in each leg. It is an array of 3 float values that are responsible for the speed and position of the thigh, knee, and foot joints.

2.3. Design of the basic behaviors



```python
def liftLeg(i):
    # i = number of a leg
    msg = Leg()
    msg.knee = -kneeSpeed
    msg.foot = kneeSpeed

    if i == 0:
        while fLPos[1] > -0.85:
            pub[i].publish(msg)
    elif i == 1:
        while fRPos[1] > -0.85:
            pub[i].publish(msg)
    elif i == 2:
        while mLPos[1] > -0.85:
            pub[i].publish(msg)
    elif i == 3:
        while mRPos[1] > -0.85:
            pub[i].publish(msg)
    elif i == 4:
        while bLPos[1] > -0.85:
            pub[i].publish(msg)
    elif i == 5:
        while bRPos[1] > -0.85:
            pub[i].publish(msg)
    stopLegs()
```



```python
liftLeg(0)
while fLPos[0] >= -0.45:
    pub[0].publish(msg)
stopLegs()
legDown(0)
liftLeg(1)
while fRPos[0] >= -0.45:
    pub[1].publish(msg)
stopLegs()
legDown(1)
liftLeg(2)
while mLPos[0] >= -0.45:
    pub[2].publish(msg)
stopLegs()
legDown(2)
liftLeg(3)
while mRPos[0] >= -0.45:
    pub[3].publish(msg)
stopLegs()
legDown(3)
liftLeg(4)
while bLPos[0] >= -0.45:
    pub[4].publish(msg)
stopLegs()
legDown(4)
liftLeg(5)
while bRPos[0] >= -0.45:
    pub[5].publish(msg)
stopLegs()
legDown(5)

while fLPos[0] <= -0.1:
    pub[0].publish(msg1)
    pub[2].publish(msg1)
    pub[4].publish(msg1)
    pub[1].publish(msg1)
    pub[3].publish(msg1)
    pub[5].publish(msg1)
stopLegs()
```

The design of the basic behaviors was the most prolonged phase of the whole project. Even when the mapping algorithm was in development there were many variables to be tuned, like the length of a separate step, the friction that generates slippage, and, therefore uncertainties. As a result, the walking algorithm has been rewritten a few times to get the most reliable and repeatable result. The number of different ways to program the movement of 18 servos is huge and there are many different factors that decide which should be used. The whole programming part has been done using Python 3 with an additional rospy library. The ease of using Python with ROS dictated this choice. Nothing has to be recompiled after the code has been modified and the methods themselves are way simpler than C++ counterparts. The code development of choice has been the Visual Studio Code from Microsoft. It is an extremely diverse tool with capabilities like installing community developed plugins. Access to such a big library of additional tools including debuggers, additional languages, and document formats has been a great convenience. It also has support for ROS, it was also a great help.

Many different ways mean that there is no one right choice that will work for all applications. This is where gaits are introduced. Gait is a pattern in which the hexapod legs will move to produce a synchronized walk. There are many different iterations but the most significant are tripod gait, where only 3 legs stay in touch with the ground and the gait where there are 4 to 5 points of contact. The first one is not as stable as the second one. Tripod gait is leaving only 3 points of contact which means that if the robot swings its legs to hard it can fall over because there is only one point of contact on one of the sides. That is why it is so important for legs to not be too long or too heavy. On the other hand, the tripod gait is way easier to control. All 3 legs in the gait can get the same command from the controller to do one move at the same time, which makes it easier to program. It is also quicker. The robot can travel greater distances in one step when using tripod gait. It can repel with 3 moving legs, which means it can apply more torque than it would be possible with fewer legs and still land on the remaining 3 legs and keep the body balanced.

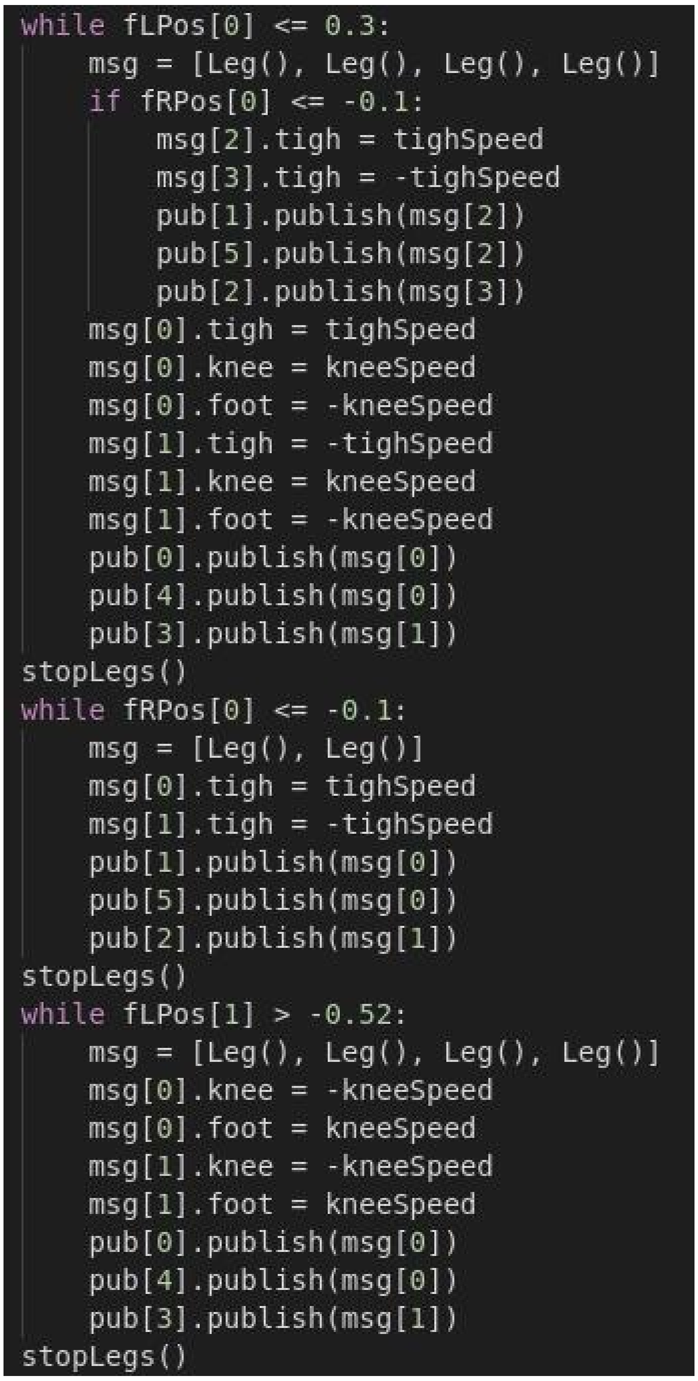
The other gait is way more stable. While keeping 4 or more points of contact with the ground robot is way more likely to stay on its legs. Moreover, it is way more likely to perform the same action every time it is executed. In comparison when tripod gait was used for turning the angle by which the robot will turn was more or less random and difficult to control. There are many different variations of this gait but the main differences are the order in which the legs are moved and the timing. That is why it is 4 or more points of contact. The timing can be set so one leg can be lifted mid-move of the other and it makes the whole action way smoother but and quicker but at the same time introduces unwanted complexity and implementation is more time consuming so it has been decided to keep five points of contact with the ground.

The final design of the robot utilizes both gaits. The one for walking is a tripod gait because when it comes to moving in a straight line precision is not as important as it is when it comes to angles and turning. Therefore when the robot is turning it is utilizing the 5-point gait for greater precision. That is incredibly helpful when creating an algorithm for mapping and localization.

2.4. Design of the mapping and localizing algorithm

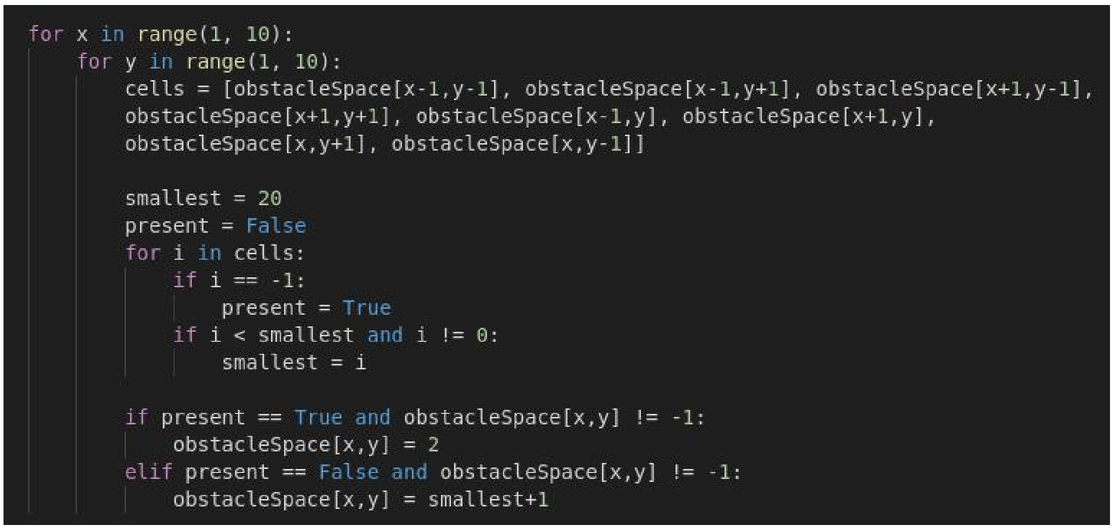
In theory, the most difficult part of the project turned out to be the most fun and easy. The mapping would not be possible without suitable sensors and a sensible representation of the world inside the robot's program. For this part, additional library NumPy has been used for the easier creation of multidimensional arrays.

Not only robot does not see the world as humans see it, but it also does not think in a way the human would. That is why the sensible representation of the real world in a robot program is one of the most difficult ideas to come up with. In this project, it has been determined for the robot to use a two-dimensional grid, where one cell is about a size of 0.25m square. The reason why being the size of the robot itself. The robot has dimensions of 22.5x33cm which means that the tightest space it can squeeze through is around 34 cm. It is also important to remember that the localizing algorithm will have an error building up as the robot is exploring, slipping, and making mistakes. Taking all that into account that the size of a grid in a cell will be the size of the robot plus a margin of error slowly building up. There is no point to mark a cell as free from obstacles when the robot can not squeeze through there because it is too wide.



```
while fLPos[0] <= 0.3:
    msg = [Leg(), Leg(), Leg(), Leg()]
    if fRPos[0] <= -0.1:
        msg[2].tigh = tighSpeed
        msg[3].tigh = -tighSpeed
        pub[1].publish(msg[2])
        pub[5].publish(msg[2])
        pub[2].publish(msg[3])
    msg[0].tigh = tighSpeed
    msg[0].knee = kneeSpeed
    msg[0].foot = -kneeSpeed
    msg[1].tigh = -tighSpeed
    msg[1].knee = kneeSpeed
    msg[1].foot = -kneeSpeed
    pub[0].publish(msg[0])
    pub[4].publish(msg[0])
    pub[3].publish(msg[1])
stopLegs()
while fRPos[0] <= -0.1:
    msg = [Leg(), Leg()]
    msg[0].tigh = tighSpeed
    msg[1].tigh = -tighSpeed
    pub[1].publish(msg[0])
    pub[5].publish(msg[0])
    pub[2].publish(msg[1])
stopLegs()
while fLPos[1] > -0.52:
    msg = [Leg(), Leg(), Leg(), Leg()]
    msg[0].knee = -kneeSpeed
    msg[0].foot = kneeSpeed
    msg[1].knee = -kneeSpeed
    msg[1].foot = kneeSpeed
    pub[0].publish(msg[0])
    pub[4].publish(msg[0])
    pub[3].publish(msg[1])
stopLegs()
```

The robot has an idea of a direction in which it is going in reliance on the absolute directions - north, east, west, and south. This means it can only go in those four directions. As the robot is walking towards a target it is determined that in five steps robot should travel around 0.5m so after five steps forward the algorithm will change current coordinates to the new cell respectively. It will be a different cell depending on the direction the robot is facing. When the robot is turning it should always turn towards one of the basic directions, that is why it was so important to get degrees of turn right. The four sensors around the body are there scanning in search of obstacles that are around the robot and in front of it. If one of the sensors will detect an obstacle it is placed on the internal map and cells occupied by the object are marked as taken and not possible to visit. After that is done, the algorithm will translate the map with obstacles to the map of repulsive fields generated by the obstacles. The algorithm will look at all the neighboring cells of the current cell and if any of them is marked as "-1", which means there is an obstacle there the current cell will be marked as 2, which is possible to visit, but it has the lowest "mark". Every cell that is not marked as "-1" or "2" will be marked as n+1, where n is the smallest value assigned to the neighboring cells. This way the algorithm can pick the safest paths, just by preferring the cells with the highest "mark". When the map has been produced the robot will plot the way around the map. The algorithm will first try to go in the direction of x coordinate if it is not possible it will try to go in a y-direction. When that is not possible then it will try the opposite x and y directions. For example when it is not possible to go north and east, then it will try south and west. This and a restriction that the algorithm will never visit the same cell twice infer an issue. It is possible for the algorithm to get stuck when it will cut itself out from the exit. It can be fixed in two ways First would be writing a second algorithm that would prioritize the y-direction first, then just pick the shortest one. That would, however, take twice as much time and computing power. A better fix would be allowing the algorithm to go back to the visited cells, then find the places where the lines cross and cut out the unnecessarily visited cells. This way the run of the algorithm would be longer, but at the same time, a path will always be found if there is one.



```
for x in range(1, 10):
    for y in range(1, 10):
        cells = [obstacleSpace[x-1,y-1], obstacleSpace[x-1,y+1], obstacleSpace[x+1,y-1],
        obstacleSpace[x+1,y+1], obstacleSpace[x-1,y], obstacleSpace[x+1,y],
        obstacleSpace[x,y+1], obstacleSpace[x,y-1]]

        smallest = 20
        present = False
        for i in cells:
            if i == -1:
                present = True
            if i < smallest and i != 0:
                smallest = i

        if present == True and obstacleSpace[x,y] != -1:
            obstacleSpace[x,y] = 2
        elif present == False and obstacleSpace[x,y] != -1:
            obstacleSpace[x,y] = smallest+1
```

3.Implementation

For any computer science project, it is common that many of its initial assumptions or even all of them are subject to change. Changes are made under the influence of new findings, new technologies, customers, or some of the initial requirements that can not be met in time or at all. This type of project is especially prone to such changes as new knowledge is being acquired on a daily basis and many things have to be simplified or expanded significantly.

One such aspect is the design of the robot itself. The first designs were assuming an equilateral hexagonal body with 6 legs, each perpendicular to the center of the body. However, this design had one significant downside. It would be way more difficult to control legs to walk in a straight line, while only middle legs are perpendicular to the direction of robot motion. For this reason, the shape of the robot's body changed significantly. Instead of symmetric hexagonal body robot from now on has a rectangular body, with the length of a body almost twice as long the width.

When the time to assemble the robot came new issues have emerged. CoppeliaSim only supports triangular meshes in a format that was not standard for Autocad 2020. It took some time and conversions to finally import all the meshes inside the simulation. Then it was necessary to convert the meshes to convex shapes which helped a lot with the simulation itself. The convex shapes are objects optimized for collision detection. Without the conversion, none of the physics engines integrated into CoppeliaSim was able to sensibly simulate the model in a way as it was intended to work.

Soon after that, another problem with the design of the robot became clear. The legs are getting into each other way without making any significant motions. There was only one solution. It was necessary to go back to CAD designs. After some trial and error, it has been decided to create a body of a length over twice as long as wide, with final robot dimensions of 33 cm wide and 22.5 cm long. This length of a body in relation to legs works great while still being compact and balanced enough to not fall over when performing a tripod gait walk.

Right before the mid-project presentation, it was time to show a working prototype of the project. It was important to create a simple but efficient algorithm that would control the robot's walk and show that it can go through a corridor without bumping into walls. That is when the first version of the walking and turning behaviors has emerged. Also, the two front sensors have been added. The first version of the walking algorithm did not use either kinematics or positions of the motors. The first version was based purely on velocities and time. Each motor has been told to move with fixed velocity in the fixed time. The main issue of this solution was a random position at which each motor could stop. If we control only the time and speed at which the servo is turning we are forgetting about friction inside the motors, which will change depending on the velocity the moment before we give a new command. If the engine was in motion before we gave it a command it will meet less friction, than in case when it would start from a complete stop. Not only that, the weight of the robot itself and gravity will also impact the speed of servos. The remedy would be using joint positions instead of velocities to determine where to stop the motor. Moveit package from ROS seemed to be the answer. However, it needed a URDF file, which is a textual description of a robot in an XML format. After it has been produced it was possible to generate a moveit package and using it program behaviors like walking using poses and generated math. Unfortunately, it turned out that the package generated by moveit is not compatible with all the work that has been done until then, so the decision had to be made. Continue discarding all the previous work and use kinematics or roll back all the kinematics designs and go back to basic, manual control of motors. The decision was made to roll back the kinematics design. The final version of the walk is using encoders to determine the position of each motor and based on that plot at which point it should be stopped.

The "turning" action was developed in parallel to the "walk" action. At first, both were utilizing the same tripod gait. Although, with the development of mapping algorithm turning had to be more precise and stable than tripod gait could ever be. That is why the 5-point gait has been used. It still is not perfectly reliable, but in robotics precision and repetitiveness are something that has to be achieved, not something gave. That is why motors are still actuated with the use of positions of the motors. That gives the most certainty that can be achieved.

The implementation of the mapping algorithm was most of the time unproblematic. The robot is using a two-dimensional array of zeros. The array is resizable, there are no constraints on the size of the array itself, but the cells are determined to have the area of 0.25m^2. It is significant because the whole algorithm controlling the path of the robot is set to think that 5 calls of the "walk" action mean moving one cell ahead which approximately is 0.5m. However, it is very easy to change those values simply by changing the speed at which the motors are moving or by adding or deleting calls of the "walk" action. The placement of detected obstacles on the map is also quite simple. Sensors are outputting the distance to an obstacle, knowing the position of the robot on the map, a direction which it is facing all that is left to do is place the obstacle on the map in the appropriate cell. The biggest issue is the positioning of the robot itself. It is incredibly difficult to precisely track the robot's location in the real world and place it on the map. The task is complicated because of the nature of the movement. The forward motion of the whole robot is composed of moves of 18 different servos instead of at least one in case of wheeled systems. This means that the velocities of motors and a robot are two different things. One can be more or less precisely plotted from the other but with uncertainty greater than it is worth. The issue of tracking robot’s localization has been minimized by thorough testing, which provided a good approximation of the speed at which the robot is walking and the distance the robot can travel with one step.

4.Testing

In robotics, it is important to remember that there are a few types of tests that have to be performed. The first type of test that is conducted is the test of all sensors and actuators. It is important to test each piece because it is the only way one can find out about imperfections each sensor and actuator has. The internal friction of motors, the margin of error of a sensor. Those things will be distinct for each piece of hardware and without finding out about them it would be incredibly difficult to accurately plot the position of the robot. The next kind of testing that has to be done is the testing of simple actions like walking and turning. How reliable is it and how many calls of a function will it take to travel 1 meter in a straight line. Other tests will include checking if the path planning algorithm can plot the right path and when it can not do it.

It is worth noting that not only normal behavior has to be tested but also exceptional behavior. This way one can find out about the bugs that would stay hidden the other way.

The tests documented in the test specification are tests constructed to perform on the final product and designed to present its capabilities and functions.

The environment in which the tests are conducted is an important aspect. The tests for the purpose of this project have been conducted on a machine running Pop!_OS 18.04 LTS based on Ubuntu 18.04. The simulation environment: CoppeliaSim Edu V4.0.0 rev4, with Newton dynamics engine on default settings.

4.1. Overall Approach to Testing

For this project testing has been conducted as a natural part of development. They were necessary to find out about the information the sensors are returning and how motors behave when actuated. It was also necessary to test behaviors like walking, turning, and avoiding obstacles. The best way to see if newly developed behaviour works are tests. Finally, the testing of the mapping and localizing algorithm has to be done.

Due to the specific nature of the project, the testing has been done during the development of each phase and then the phase was finalized with the final test that would ensure that most of the bugs have been fixed and the state in which the current phase is right now is satisfactory. This way the risk of some parts of the project not being done or uneven quality is minimized. It is natural that after the final test each phase has been further modified and developed. Sometimes different parts of the project can influence each other.

4.2. Testing of sensors and motors

The upside of using simulation is the fact that sensors and motors do not have the distinct imperfections that real-life devices introduce. However, they still have their own parameters that have to be investigated. For the first model, it was not necessary to test the sensors, because both were producing only boolean values and the range was set to 0.5m. Those things do not apply to the proximity sensors used in the final version. To test the sensors the robot has been placed in 3 positions in relation to the object that can be seen by a sensor. First, it was 200, 300, then 400. This way it became clear that the sensor is returning the floating-point value 0.n, where n is the distance in mm measured by a sensor. The result of this experiment is evident in the mapping function. It is using distance to the object to more accurately mark obstacles on the map.

The tests of motors were more complicated. As it would not be beneficial to test each motor individually, they were tested along with the development of walking action.

4.3. Testing of walking and turning

To find out about different qualities of programmed walking action it has to be tested. To test it the robot had to walk along 3 different objects. Each different length - 0.5m, 1m, 2m respectively. The distance that has been traveled has been measured by calculating the distance of [x,y] coordinates of the base of the model before and after conducting a walk. This way it was possible to measure distance traveled very accurately. Moreover, this way it was possible the precise deviation from a straight walk. If the robot started the walk at coordinates [x,y], and was walking along x, then any change in y coordinate will indicate how imperfect the walk is. The test indicated that the robot is walking around 0.016m/s and it needs around 5 calls of a walk() function to travel 0.5 meters.

The turning function was tested in a similar way. The robot was told to execute the spinRight() function a number of times. To calculate the difference in rotation of the robot the [x,y] coordinates of the tip of the robot's foot before and after performing a spin have been noted. Now the distance from the center of the robot to the tip is the radius of the circle and [x,y] coordinates before and after a turn are determining a sector of a circle. Now all that has to be done is to calculate the angle drown by connecting [x,y] coordinates to the center. That is the angle of turn.

The result of the tests is now used in the localization algorithm. The robot will approximate its position based on the number of steps it has made. Similarly, the turning is tracked by counting the number of times the spin function has been executed.

4.4. Testing of the mapping and localizing

Most of this process took place in separation from the main program. It is distinct enough because while the development of the walking, turning, or sensing behaviors is impossible without some representation of the physical model, the development and testing of localization can be conducted away from the model and other parts of the project. This enabled the parallel development of mapping and localizing algorithm and the rest of the project. This means that the prototype of the final algorithm from go_to_target.py can be found in find_path.py.

To test the algorithm used for translating the map, the positions of obstacles had to be simulated. Coordinates have been set on the map just as the robot would do it. Then the map would be translated and printed. The test would be passed if the map would be produced where obstacles are left as they were marked, all cells around them were marked with "2" and the rest of the cells was marked with n+1, where n is the lowest value of all the neighboring cells.

To test the algorithm used for finding the path it was important to think about the logic that is making the decisions. For example, there is no point in testing if paths are generated each time, because by nature, in some cases the path cannot be found. On the other hand, it would be wise to check if the path can be found in both directions. From start go goal, and from goal to start. This is inferred by the most significant statements (if xt <symbol> x and yt <symbol> y). By making it find a path from each corner of the map to each corner we test each of the statements and fail in doing so will clearly indicate which of the statements is failing.

5.Critical Evaluation

The goals of this project were very clear from the beginning. Develop a hexapod robotic system that would be able to move around the environment, map it, and plot a path from one point to another. All of them have been addressed in the design and applied in the final version of the robot. Although, there are some issues that should be fixed.

The model of the robot in CoppeliaSim is working but a not very detailed simulation. While the 3D models created in CAD software have a higher level of detail, the model inside a simulation is built out of convex shapes. They are simplified for a simulation to run more smoothly, but if the CAD meshes were done better there would not be a need for conversion to convex shapes. The other issue is the reliability of the algorithm for walking. The robot will never walk in a completely straight line. It is caused by the rate at which the program is looping commands and the speed of motors. The rate is causing delays in reading data from sensors and then sending commands to actuate motors. The speed of the servos can also influence the reliability of the algorithm. The faster the electric motor is turning the bigger error is introduced when stopping the motion. Moreover, both walking and turning algorithms are lacking in smoothness. It is caused by a simplistic design of the algorithms. Within the available time it was more beneficial to write a simple but working algorithm instead of something very polished and natural-looking, but very time-consuming. The next issue would be the accuracy of the algorithm that tracks the motion of the robot on the 2D map. Due to not completely reliable algorithms for walking and turning the tracking is also lacking. The longer robot runs the bigger the error and the solution for it might be a better set of sensors not only to see obstacles around the robot but also to correct the error that walking introduces. For example, with the depth camera, the robot could measure its own speed by comparing the change of position of certain points in the point cloud. Also, the algorithm that translates the map that robot is creating to the map of force fields has a defect that makes the robot always walk closer to the lower left side of the map. The issue that is causing this is the fact that the map is being translated only once from the top left variable [0,0] to the bottom right [11,11]. This means that the algorithm that is processing the map doesn't change the variable in the row above so fields are only generated in the cells nearest to the obstacle and radiate from the bottom right side of it. More suitable tools could be used. It seems to be a better choice to use tools already provided with ROS. While work with CoppeliaSim was overall a pleasant experience, the combination with ROS introduced many additional and unnecessary problems like the requirement of recompiling the CoppeliaSim plugin to work with custom messages. Another example would be the requirement of scripting the robot inside the simulation. ROS and Gazebo do not require such things, therefore it seems clear that using CoppeliaSim alone, or ROS with Gazebo and R-Viz would be a better option.

However, there is a large amount of work that has been done. The parts of the robot have been created in CAD software from the ground up. Then assembled in the simulation with precision. The model itself is a good enough representation of the physical model that, with some adjustments, can be built. The first two goals have been completed. Then the robot has been programmed to walk, turn, and avoid obstacles. The program is working without any bugs that could be found. The robot is also capable of mapping the environment, plotting its own way around. It can generate a map of force fields based on the information sensed in real-time by 4 sensors around its body. The goal of an autonomous hexapod robotic system has been achieved. The project consumed a great amount of time and research to produce work that I am happy with.

There are a few more ways in which this project could be expanded. It is a hexapod for a reason. The reason is the fact that systems like that are very stable due to a large number of points of contact with the ground, while at the same time being able to traverse uneven and rough terrain. Contrarily wheeled systems while easy to control do not do well in sand, or stones. This means that this project can be expanded with the ability to climb and map the environment in 3D. This way it can be used to traverse anywhere where it is too dangerous or too tight for a human. It can traverse crumbled buildings looking for survivors, explore and map caves that are out of reach for human, or equipped with RGB camera could crawl into pipes looking for leakage. The algorithm used for finding a path is a Breadth-First Search algorithm which means it might be slow when used with a larger amount of data. A better idea would be the implementation of something like A* algorithm that would be a way better way to go.

6. Annotated Bibliography

1. Neil Taylor, “MMP_S08 Project Report and Technical Work”, 2019 (Online) Available at: http://blackboard.aber.ac.uk/ Accessed 19th February 2019.
   A document that outlines information about the marking guide for the Project Report and Technical Work. This is published in the Resources folder on Blackboard.
2. Retrieved May 11, 2020, from https://www.coppeliarobotics.com/helpFiles/index.html
3. Latombe, J.-C. (2010). Robot motion planning. Boston: Kluwer.
4. Nehmzow, U. (2003). Mobile robotics: a practical introduction. New York: Springer.
5. Siciliano, B. (2010). Robotics: modelling, planning and control. London: Springer.
6. Wiki. Retrieved May 11, 2020, from https://wiki.ros.org/
7. Xie, M. (2008). Fundamentals of robotics linking perception to action. Singapore: World Scientific Pub.

7. Appendices

A. Third-Party Code and Libraries

Python numpy library - Used for easier creation of multidimensional matrices. Version 1.18.4 was used, it is an open source.

CoppeliaSim Edu V4.0.0 rev4 - For simulation purposes. It is free for students, universities and non-profit projects. Developed by Coppelia Robotics.

ROS Melodic Morena LTS - Middleware between simulation and the controlling code.

Visual Studio Code version 1.45 - Open source developed by Microsoft code editor. Capabilities can be extended with plugins.

ROS extension - plugin for VS Code, version 0.6.3. It helps to write and compile Python and C++ files for the purpose of use with ROS. The extension is developed and maintained by Microsoft.

B. Ethics Submission

Image not found or type unknown

07/05/2020

For your information, please find below a copy of your recently completed online ethics assessment

Next steps

Please refer to the email accompanying this attachment for details on the correct ethical approval route for this project. You should also review the content below for any ethical issues which have been flagged for your attention

Staff research - if you have completed this assessment for a grant application, you are not required to obtain approval until you have received confirmation that the grant has been awarded.

Please remember that collection must not commence until approval has been confirmed.

In case of any further queries, please visit www.aber.ac.uk/ethics or contact ethics@aber.ac.uk quoting reference number **15921**.

Assesment Details

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
osl1@aber.ac.uk

Full Name
Oskar Lipienski

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Development of hexapod bioinspired systems

Proposed Start Date
27th January 2020

Proposed Completion Date
11th June 2020

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Design, and programming a 6-legged robotic system that can map the enviroment and find its way around it. Project is involving ROS, CoppeliaSim and AutoCAD software

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

Is your research study related to COVID-19?
No

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here: