Multi Robot Task Simulation
(In MATLAB Software)

Kirata Iotam
Student Id No.: S11045193

Krishneel
Prasad
Student Id No.: S11050297

School of Engineering and Physics
Faculty of Science and Technology
University of the South Pacific

November 2011.

Supervisor:
Dr. Praneel Chand
School of Engineering and Physics
Faculty of Science and Technology
University of the South Pacific

A report submitted in fulfillment of the requirements for the degree of Bachelor of Engineering Technology.

Acknowledgments

Firstly, we would like to thank our project supervisor, Dr. Praneel Chand for giving us the opportunity to work with this project. Also sincere thanks go to him for his constant guidance, enthusiasm and support throughout this project. We would also like to acknowledge and thank the following people for their help in any way in this project.

- Mr. Robert Li
  (Electrical/Electronics Lab Technician, Department of Engineering)

- Mr. Ravinesh Singh
  (Senior Lecturer: Electrical/Electronics, Department of Engineering)

- Mr. Sumesh Narayan
  (Lecturer: Mechnical/Manufacturing, Department of Engineering)

- All our colleagues and fellow students in the EE300 class of 2011 deserve earnest thanks for their camaraderie and help.

- And finally our Family Members for invaluable support throughout the semester.

Declaration of Originality

We hereby declare that the work presented in this report, is to the best of our knowledge and belief original, except as acknowledged in the text, and that the material has not been submitted previously, either in whole or in part, for a degree at this or any other institution.

Kirata Iotam	Krishneel Prasad
(Student Id No. S1104)	(Student Id No. S11050297)

November, 2011.

Abstract

Technology is a never-ending process. This project involves using Matrix Laboratory (MATAB) to simulate a Multi Robot task system. This involves coding in M-Files to create Robots, moveable objects and even obstacles. If this concept was to be implemented in real life applications, it could be used to diffuse bombs, transporting hazardous materials and so many other applications. Therefore this will be beneficial to the lives of others is a huge contribution to the community. The project's aim is to enable robots to initially approach the object and then move it towards the goal location

Firstly, our first objective was to construct a movable object. Secondly, to implement and simplify models of force exertion by applying collective behavior methods on model of force strategy. For this motion of rigid bodies was taken into account. Using several robots that cooperate in a group has several advantages over solving the same task with one single robot

Redundancy- if one robot gets stuck or destroyed, the group as a whole can still function but performance may be reduced.

Simplicity - instead of loading just one robot with the task (including advanced sensors) the tasks can be performed by several smaller and simpler robots. Reducing performance load and cheaper if goes into production.

Flexibility – Groups of simple robots can be easily rearranged or partitioned into small subgroups to adapt to new environments or functions. There were two types of motion included in the matlab program. Firstly, Path Planning when the robot uses this method of motion it can move directly towards goal, Circumnavigate obstacle (if any) and variations of these are also possible. But the robot needs to know the location of the goal or target. Secondly, when the robot uses reactive control the robot can reacts directly to current sensory information and has no memory – no look-ahead reacts to the current stimuli.

Table of Contents

Chapter 3 Forces and Torque

Chapter 4 Friction

Chapter 5 Experiments, Results and Discussion

Chapter 6 Conclusion and Recommendations

List of figures

List of tables

Chapter 1

Introduction and Literature Review

1.0 Multi-robot system overview

Multi robot system has recently become an interested area in robotics application. Hence researches invest in this area their technical knowledge to solve such applications which are useful in many areas. One of these applications is pushing an object to the ultimate target by one or more robots. This paper is based on the Multi Robot system simulated in Matlab software. There are number of approaches that have been exercised to carry out this task successfully. The following section will examine different avenues toward pushing task system.

1.1 What is Robotics?

A system that contains sensors, control systems, manipulators, power supplies and software all working together to perform a task. Designing, building, programming and testing robots is a combination of physics, mechanical engineering, electrical engineering, structural engineering, mathematics and computing.

1.1.1 Definition of Multi-robot system and Pushing Strategy

Multi robot systems guaranteed improved performance and increase the probability of solving large scale problems [1]. Generally, multi-robot has some remarkable advantages over a single robot system such as larger range of task domain, greater efficiency, improved system performance, robustness, ease of development and most of all lower economic cost [2]. The involvement of multi-robot in pushing objects has become one of the main objectives of many researchers in nowadays. The control of manipulators (box pushers) on the manipulating objects is challenging due to the nonlinearities equations of motion [3]. Apart from this, the implementation of multi-robot task is dynamic such as it could be developed in Java or Matrix Laboratory (Matlab). Although it is dynamic in software, approaches toward pushing box task execution in wheeled mobile robot may totally different such as one may use A* algorithm approach while other software such matlab use Multiple Impedance Control law or Behavior based approach. Many avenues toward multi robot object pushing have been developed and had physically tested on hardware for their efficiency on real life situations. Some of these approaches are Behavior base approach, Virtual structure and Genetic algorithm approach. In contrast, the important similarity out of the two methods is the need of force modeling on an object (manipulating object) which known as vector force composition [3]. Modeling of forces on the object is important for the movement of the object with respect to the robot position and also to give a close realistic perspective to real life engineering problems [4]

1.2 Literature review

Part (I) Task Allocation

1.2.1 Multi-Robot System

In Liu and Wu (2001), the basic selection of the robot is mainly on the fitness function. To add, for the initial selection of robots, the fitness of each robot is examined and compared with the given task [2]. The given task in this context is pushing an object, however the only thing that matter in the task is the amount of force required to push the object. For instance; if the object has a mass larger enough to create larger friction force, two robots has to do the task if they exceed the demands of friction force. Otherwise, one robot sends to do or execute the given task. Thus, the selection of the robots is base on their current level of fitness to carry out the task successfully.

1.2.2 Q- Learning algorithm in a team of cooperative multi-robot task

In this approach, Wang and Silva applied the use of multi-robot cooperative strategy in game their plan [1]. The algorithm they used is base on Q-learning algorithm which was applied to push a box by 3 robots. Since the box has finite number of contacts, they limit the number of contacts areas between the robot and the box into six contacts points, illustrated in the diagram below.

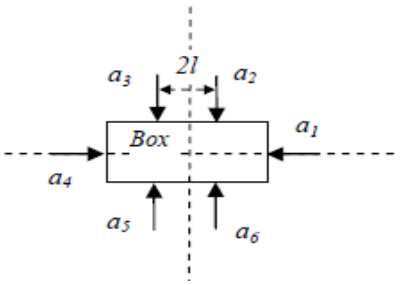


Figure 3: $a_1 – a_6$ are specified contacts points between the robot and the object (Source: Wang and Silva, 2003)

Part (II) Task Execution

1.2.3 Genghis II six legged robots

Unlike ordinary wheeled robot, Genghis II is a six legged robot. It has used to push an object to a target location.

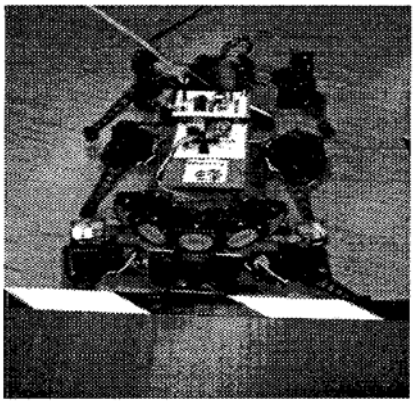


Figure 1a: Genghis II six legged robots

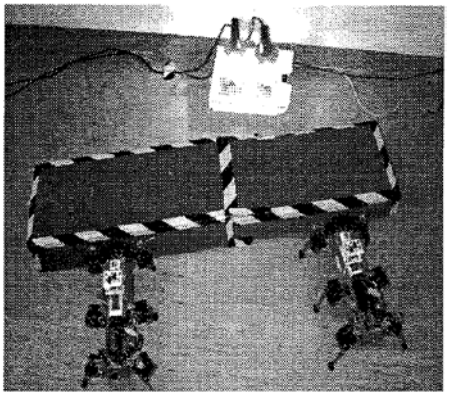


Figure 1b: Two Genghis II six legged pushing a box

(Source: Kristian.T.S atal, 2003)

According to Kristian.T.S atal works they construct Genghis II with two front sensors called whiskers and pyros which are located at the front of the robot [3]. The whiskers sensors (placed at the right and left of the robot) are responsible for correcting the position of the robot in such a way that the robot should perpendicularly in contact with the object (box). If the robot diverted from the target position the sensor feedback from either left or right made the robot to adjust itself to the right position. The second sensor (pyros, used to detect the moving edge of the object) measure the reflected light bounced back from the object. The most reflected light is the direction of the robot that should take. For example; if most of the light came from the right side, then the robot moves to the right, otherwise left or moving straight if light density on either side of the robot had similar density. These movement caused by the robot was also caused the object to move in a same way.

The methods applies in this situation was more on reactive control with radio as a means of communication between robots. The approach was basically taking turn approach where each robot takes it turn first by moving an object then aligned itself and finally send message to it peer, a message “your turn”.
Modelling of forces is not important in this approach. In addition dynamic modelling of the robot movement with respect to the object was not mathematically modelled. The reason is that sensors governed most of the control system where they were used to determine the robot state at any time.

Communication is vital in this approach, without communication the robot tends undid the other robot work. As a result two robots can drive to object to out of bound or abandoned it [1].

1.2.4 Object manipulation by three robots base on move and constrain approach

In this approach, Ahmadabadi and Nakano constructed four robots to manipulate an object using a move and constrain approach [4]. They applied this approach on four robots called Ichiro, Jiro, Saburo, and Shiro to pick up an object and carried it to the target location. Although this approach was beyond the project objectives, pushing strategies exercised in this paper will only discussed and examined.

`

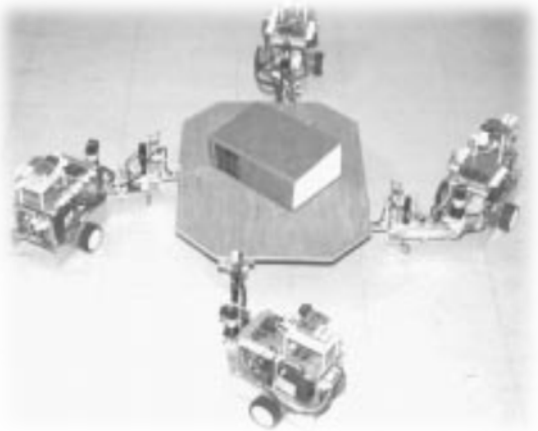


Figure 5: Ichiro, Jiro, Saburo, and Shiro
(Source: Ahmadabadi and Nakano, 2001)

One of the important issues toward the implementation of pushing strategy is considering external forces exerted by the robots on the object [4]. In addition, Ahmadabadi and Nakano believed that the mass of the object is concentrated at the centre of the object. From this, they were able to derive equations to give a clear understanding on how the object moved with respect to the robots' interactions in mathematical terms. These two major equations were shown below, which were the basis of this approach.

Move equation:

$$\sum_{i=0}^{n}(r_i \ \ x \ \ F_i + T_i) \ = I(\ \ddot{\theta}_{desired} + \ C_r \ \dot{e}_r + K_r e_r\ ) \text{ [4]}$$

……equation 3

$e_r = \theta_{desired}$ - θ, C_r – rotational damping, K_r – spring coefficients, I – moment of inertia at the centre of gravity

Constrain equation:

$$\sum_{i=0}^{n} F_i = M(\boldsymbol{Q} + \boldsymbol{C}_P \dot{e_d} + \boldsymbol{K}_p e_d) \ \text{ [4]}$$

……equation 4

$\boldsymbol{Q} = [-\boldsymbol{\omega}\ \boldsymbol{x}\ \boldsymbol{\omega}\ \boldsymbol{x}\ \boldsymbol{l} \quad -\boldsymbol{\alpha}\ \boldsymbol{x}\ \boldsymbol{l}]^{\boldsymbol{T}}$ - The matrix is developed as the basis **of Euler equation of motion** where he considered the **angular acceleration and angular velocity** of an object as a **result of the external net torque** applied [4]

$\boldsymbol{e_d} = [\boldsymbol{R_{d0x}} - \boldsymbol{R_{dx}} \quad \boldsymbol{R_{d0y}} - \boldsymbol{R_{dy}}]^{\boldsymbol{T}}$

K_p and C_p are the damping and stiffness coefficient respectively which represented as a 2x2 diagonal matrices

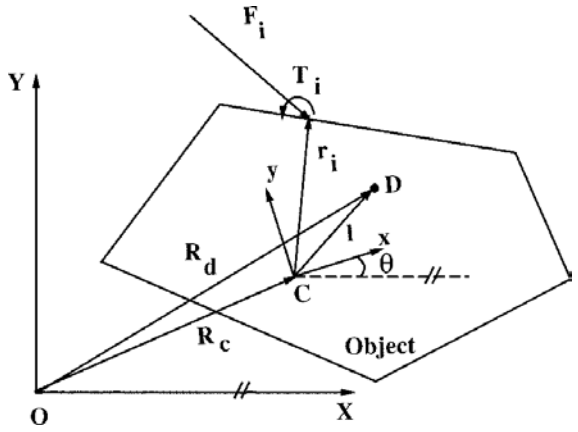


Figure 5: Simple model of forces acting on an object in 2D diagram
(Source: Ahmadabadi and Nakano, 2001)

As shown in Figure 5, $\mathbf{F_i}$ is the resultant force of (i) robots while $\mathbf{T_i}$ is the torque denoted by (i) robots. Also, $\boldsymbol{r}_i$ represents the position of *i*th robot contact point in the object coordinate system [4].

1.2.5 Formation control applied on cooperative robots for pushing task

In this paper, Abbaspour and Moosavian were constructing cooperative wheeled mobile robots that can manipulate (push) an object from its initial position to the final location. Additionally, by using multiple impedance control (MIC), they were able to control the interaction of forces between the robot and the objects [7]. Moreover, this paper also considered navigation system which the robot used while pushing the object to its target location. However, navigation system is beyond the objectives of this project, therefore the parts such as moving the object and controlling the interactions of forces will be the basis of discussion in this section.

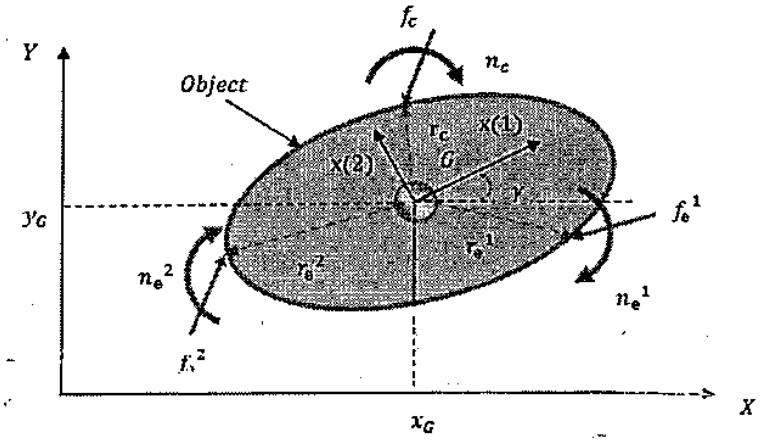


Figure 5: Model of forces and torques applied on an object in 2D
(Source: Abbaspour and Moosavian, p4)

One of the important aspects for considering the movements of the object is to model forces acting on it by number of robots. In the combination of Newton second law with Euler equation of motion, forces and torques can be modelled accordingly [7].

Sum of forces (Newton 2nd Law of motion)

$$f_e + f_0 + f_e^1 + f_e^2 + m_{obj}g = m_{obj}a_G \text{ [4]..... equation 5}$$

$$g - gravity, a_G - \ acceleration\ of\ centre\ of\ gravity$$

Torque equation (Euler equation)

$$n_c + n_o + r_c x\, f_c + r_e^1\, x\, f_e^1 + r_e^2\, x\, f_e^2 + n_e^1 + n_e^2 = I_G\, x\, \dot{\omega} + \omega\, x\, I_G\omega \text{ [4].....equation 6}$$

$$\tau = I_G\, x\, \dot{\omega} + \omega\, x\, I_G\omega \text{ [7]}$$

Euler equation of motion considered the **angular acceleration and angular velocity** of an object as a **result of the external net torque** applied [7]

Another important part of this approach is controlling the interaction of forces exerted by the robots on the object. As highlighted by these scholars, multiple impedance law is preferred for its accuracy in manipulating an object by enforcing designated impedance on all cooperating manipulators and the manipulated object.

Vector of generalized forces:

$$Q^{(i)} = Q_{app}^{(i)} + Q_{react}^{(i)}$$

$Q_{react}^{(i)}$ – Effect cause by the reaction load on the i^{th} robot
$Q_{app}^{(i)}$ - Controlled torque by the robot, divided into two parts which are motion concerned and force concerned.

$$Q_{app}^{(i)} = Q_m^{(i)} + Q_F^{(i)}$$

$Q_m^{(i)}$ - Motion concerned is amount of controlled torque that caused the movement of the object

$Q_F^{(i)}$ – The force required to take into account the effects of the reaction load

1.2.6 Basics concept of Multiple Impedance Control

The force for each robot is based on the concept of the Impedance control [7]. To add, the two major components of MIC and these were used in this project:

Position control strategies – this happens when an end effector position (x_2) or the object position (x_3) are provided with better tracking path to the goal location.

Force Regulation control – the force of end-effector is controlled based on the acceptable trajectory of the object and presuming that the object is a rigid body. In this case, x_2 is not controlled, F_e could provide tracking errors of x_3:

$$F_e = b_2(\dot{x}_2 - \dot{x}_3) + k_2(x_2 - x_3)$$

b_2 - damping coefficient (N. s^2/ m)
k_2 - Stiffness coefficient (N/m)

Considering the equation of the moving object:

$$m_3\ddot{x}_3 = f_0(x_3.\dot{x}_3) + F_e + F_c$$

m_3 - Mass of the object
f_0 - All potential frictional forces
F_c – contact forces acting on the object

Therefore the approximate controlled force and the error function are as shown below:

$$F_{edes} = m_3\ddot{x}_{3des} - f_o(x_{3des}, \dot{x}_{3des})$$

$$e_f = m_3(\ddot{x}_{3des} - \ddot{x}_3) - f_0(x_{3des}, \dot{x}_{3des}) + f_0(x_3, \dot{x}_3) + f_c$$

To provide a better controlled force on the object e_f must be equal to zero

1.2.6 Applying Repulsive Forces for collective box-pushing

The approach by Liu and Wu shows a pushing box strategy which basically on applying repulsive of forces. They asserted that Hooke's law is necessary in modelling this application. They stated that the group of robots can push the box if and only if their distances to the box are less or equal to the defined constant Io [2]. The force for each robot is calculated according the formula given below.

$$F = \begin{cases} n.(Io - di), & if\ di \leq lo, \\ 0, & otherwise, \end{cases} \qquad \textit{Equation 1}$$

di – Euclidean distance

n - Positive coefficient

Io - Range of the box

Moreover, they also showed that the net force applied on the box by the robots is measured as the displacement of box movement and the direction of the movement.

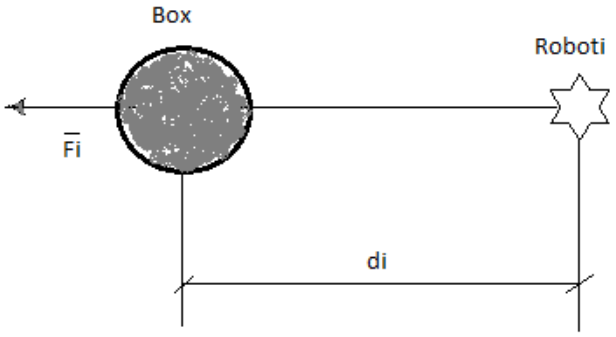


Figure 4: The plan elevation of a group of robots (i) with the distance of di and the box predefined Io radius (shaded area)

1.3 Conceptualization

1.3.1 Task Allocation

From the lists of approaches discussed earlier, selected modelled has to be examined. The

initial selection of the robots is base on the fitness function; specifically the robots are selected to execute the task base on the object characteristics. Additionally, the only factor which differentiates each object is their masses. Every object will have their one unique masses and this lead to the selection of the robots. For example; if one object weighted 5 kg, and for simplicity the friction force is calculated to be 5 N. This amount of friction force has to exceed by a single robot and if it fails two robots will send to execute the task. Therefore, the numbers of robots depend on how big and how massive is the object.

1.3.2 Task Execution

For task execution, the important factor to consider is how to control the forces applied by the group of robots. The solution seems to favour the use of Behaviour Based Approach (BBA) with Multiple Impedance control (MIC). The purpose of BBA is for calculating the force magnitude and to determine the resultant force directions while pushing the object. The MIC on other hand will take control of the contacts forces as the robots continue pushing the object.

First of all, the calculation of force is done using *Equation* 1, BBA. However, as the robot actually pushing the object, MIC takes over. This means that the calculation of force is switched from BBA to MIC if and only if the robots end-effectors are in contact with an

object. At this stage, force control is taken over by MIC which means the movements of the object must respect closely to the applied force, detailed of MIC concept explained in section 1.1.5. Therefore, the two approaches are examined here and combined to give a better control over the entire system (Multi robot task simulation)

1.4 Structure of the report

Chapter 1 gives brief introduction of the topic and also discusses and summarises the journals and other research material used as the Literature Review.

Chapter 2 discusses how a movable object is created and also how the appearance of the robot is modified in order have an arm extension.

Chapter 3 contains parts of coding and flow charts to show how force and torque is applied to the object. And also equations.

Chapter 4 provides a brief overview of how friction influenced the movement of the object and how the equations were formed.

Chapter 5 describes the tests that have been carried out at each stage of developing the

Chapter 6 contains a summary of this report and some important conclusions

Chapter 2

Creation of Moveable Object and Changes in Robot Appearance

2.0 Construction of moveable object (MovObject) in Matlab

Constructing objects (moveable objects) in programming perspective (such as Matlab software) involves the use of coordinates to label different vertexes (MovObjectVertex) and the object edges were joined using MovObjectEdges, the following figure shows the codes for constructing a moveable objects.

```
function [MovObjectVertex, MovObjectEdge] = getcoord ()

MovObjectVertex = [1, -1; 1, 1; -1, 1; -1, -1; 1, -1];

MovObjectEdge = [1, 2; 2, 3; 3, 4; 4, 1];
```

Figure 2.1: Moveable vertexes and edges

From the figure it shows that the moveable objects consists of four coordinates for each vertex (vertexes for each corner of the rectangular shape), (1,-1) for vertex 1, (1, 1) for vertex 2, (-1, 1) for vertex 3, (-1,-1) for vertex 4 respectively. However the fifth one (1,-1) is the duplicate of the first coordinate for joining the last two vertexes of the rectangular shape (Moveable object).

2.1.1 Detecting properties of a moveable object

In this section, three major changes in the program will be discussed for making the moveable object detectable by the robots; lists of m-files involved are as follows:

1. sim_obstacle_detect (m-file)
2. sim_irobstacle_range_submex(C-file)
3. gui_main (main control m-file of the program)

These modifications were carried out in this section were based on creating a new variable (MovObject) which had some properties similar to an obstacle, however the difference was that one is moveable while the other is fixed.

2.1.2 Changes in sim_obstacle_detect

The modifications made in sim_obstacle_detect m-file include the addition of new vertexes and edges of the moveable object (MovObject) to the function's lists of inputs. These inputs (MovObjectVertex and MovObjectEdge) are extracted from getcoord m-file (m-file which draws the entire layout of the object).
The locations of these inputs in the function must be similar to their corresponding locations in other m-files such as in gui_main (discussed later). The flow chart shown below shows the alterations carried out in sim_obstacle_detect m-file.

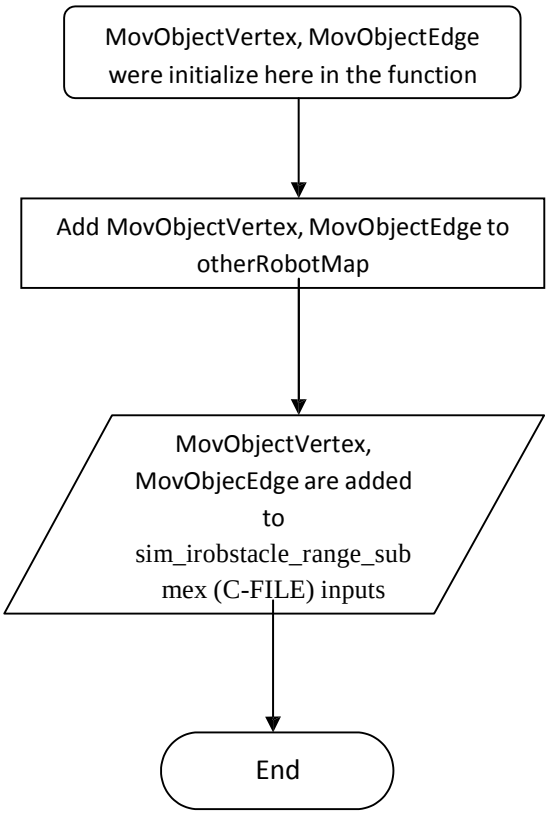


Figure 2.2: Flow chart of the added parameters of MovObject in sim_obstacle_detect (m-file)

As illustrated from the flowchart, all vertexes and edges of the moveable object are initializing first in the inputs function. Then these inputs are added to the other robot maps directories, just to be in lined with the obstacle vertexes and edges. Finally, the moveable object vertexes and edges are fed in the sim_irobstacle_range_submex (Mex function).

The flow chart clearly shows that the two inputs such as MovObjectVertex and MovObjectEdge were inserted in the same way as obstacle vertex and edges were added in the program. And this was shown in Table 2.1, the corresponding pairs between obstacles and moveable objects.

Table 2.1: The added vertexes and edges for the obstacle (Obst) and moveable object (MovObject)

Vertex and Edges inputs	
Obstacle	**MovObject**
ObstVertex	*MovObjectVertex*
ObstEdge	*MovObjectEdge*
Corresponding x and y coordinates	
Obstacle	**MovObject**
ObstVertexX	*MovObjectVertexX*
ObstVertexY	*MovObjectVertexY*
ObstEdge1	*MovObjectEdge1*
ObstEdge2	*MovObjectEdge2*

From the table, it can be seen that moveable object was added according to the obstacle parameters. As such MovObjectVertex is similar to ObstVertex while ObstEdge correspond to MovObjectEdge. Also the lists of x and y vertexes and edges for the moveable object were added similarly to obstacle vertexes and edges.

2.1.3 Modifications in sim_irobstacle_range_submex

Sim_irobstacle_range_submex function is a C-file in matlab which contains Mex function (gate way function in C – Matlab executes MEX files by finding the corresponding MEX file with the same name). This function contains coordinates transformation parameters that an obstacle has in order to be detected by any robots. Additionally, this C-file is called from sim_obstacle_detect (the calling function) with 16 inputs. As moveable objects are added, the number of the required inputs in sim_irobstacle_range_submex is increased to cater these ups. The flow chart below summarizes the changes made in the C-file.

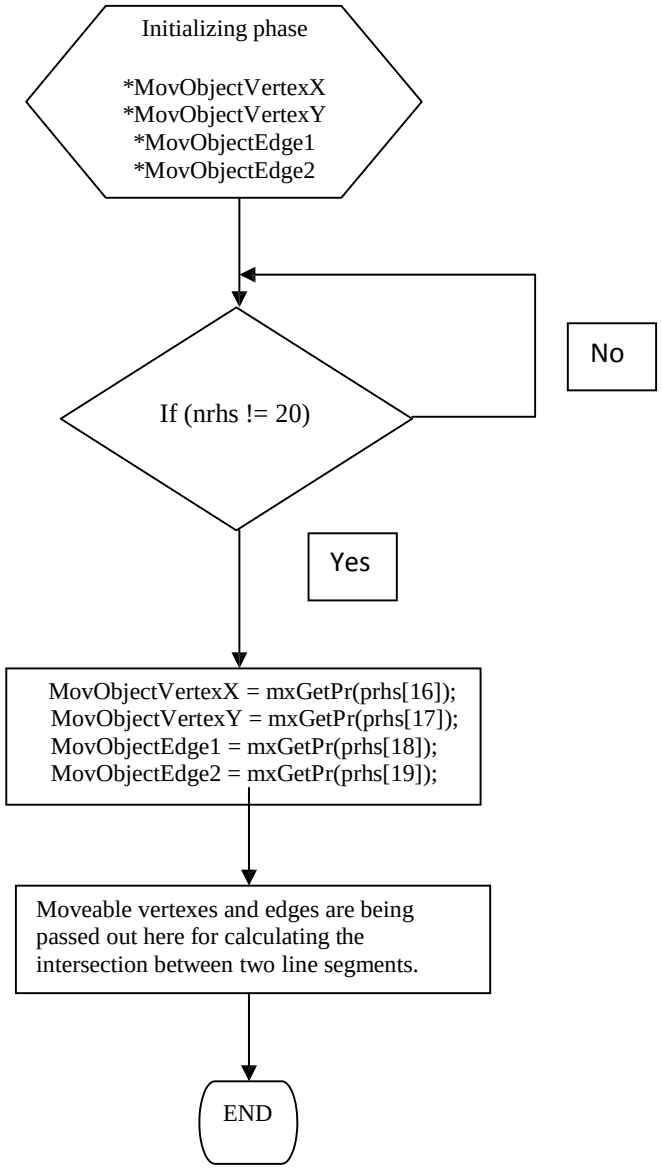


Figure 2.3: Flow chart of the added parameters of MovObject in sim_irobstacle_range_submex.c

As illustrated in **Figure 2.3**, the initializing phase consists of four pointers, **MovObjectVertexX, *MovObjectVertexY, *MovObjectEdge1 and *MovObjectEdge.* These lists of pointers were stored as **double** in the program and they originally passed from sim_obstacle_detect.m file. For this reason variables stored in the address of these inputs were easily manipulated in the C-File. In the if-else branch, lists of inputs was compared and since inputs is increase to four now from sixteen inputs, the required lists of inputs is twenty.

2.2 Modifications in gui_main

In gui_main, MovObject vertexes and edges were added as the inputs argument to get_movobject_parameters. These lists of inputs were shown below;

globalEnvData.MovObjectVertex = MovObject {allMovObjectIDs (iMovObject)}.MovObjectVertex;
globalEnvData.MovObjectEdge = MovObject {allMovObjectIDs (iMovObject)}.MovObjectEdge;

Furthermore, MovObject vertexes and edges are added to their corresponding locations in sim_obstacle_detect function in gui_main respectively.

2.2.1 Changing the local coordinate of MovObject

The changes made in sim_obstacle_detect function include the additions of specific vertexes of the MovObject to make it appear in multi-robot simulator window. The reason for these changes is to change the local address of the MovObject for it to appear in the simulator window. These changes include addition of all the MovObject vertexes to its corresponding pose current pair.

```
[MovObjectVertex (1, 1), MovObjectVertex (1, 2), MovObjectVertex (1, 3)] =
pose_add(MovObjectVertex(1,1),MovObjectVertex(1,2),MovObjectVertex(1,3),MovObjectPoseCurre
nt(1),MovObjectPoseCurrent(2),MovObjectPoseCurrent(3));

[MovObjectVertex (2, 1), MovObjectVertex (2, 2), MovObjectVertex (2, 3)] =
pose_add(MovObjectVertex(2,1),MovObjectVertex(2,2),MovObjectVertex(2,3),MovObjectPoseCurre
nt(1),MovObjectPoseCurrent(2),MovObjectPoseCurrent(3));

[MovObjectVertex (3, 1), MovObjectVertex (3, 2), MovObjectVertex (3, 3)] =
pose_add(MovObjectVertex(3,1),MovObjectVertex(3,2),MovObjectVertex(3,3),MovObjectPoseCurre
nt(1),MovObjectPoseCurrent(2),MovObjectPoseCurrent(3));

[MovObjectVertex (4, 1), MovObjectVertex (4, 2), MovObjectVertex (4, 3)] =
pose_add(MovObjectVertex(4,1),MovObjectVertex(4,2),MovObjectVertex(4,3),MovObjectPoseCurre
nt(1),MovObjectPoseCurrent(2),MovObjectPoseCurrent(3));

[MovObjectVertex (5, 1), MovObjectVertex (5, 2), MovObjectVertex (5, 3)] =
pose_add(MovObjectVertex(5,1),MovObjectVertex(5,2),MovObjectVertex(5,3),MovObjectPoseCurre
nt(1),MovObjectPoseCurrent(2),MovObjectPoseCurrent(3));
```

Figure 2.4: MovObject vertexes added to sim_obstacle_detect m-file

As for all these inputs (MovObjectVertex), pose_add function is used for transform these vertexes to the global environment.

2.3 Robot End Effector constructions

The new change that takes place in the construction of the robots is the addition of its new end effector (pushing arm of the robot). The new implementation is vital for the robot construction since to push an object the arm is acting as a safety margin to robot main body to avoid possible damages due to physical contacts in real life situations. The diagrams shown below illustrated the old model and the new model, after changes were made.

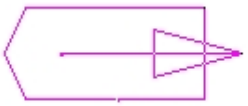


Figure 2.5a: Old design

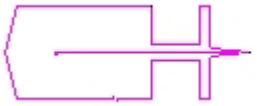


Figure 2.5b: New Design

The two types of the robots shown have similarities and differences. The old design robot has a same body shape compared to the new one. However, the differences start off with the body length; the old design has longer body length compared with the new one and of course, the inclusion of the arm for the modified one makes them more different. To note, arrow head runs from the centre of the robot are not the physical part of the robot, it was just there for direction purposes.

The new changes brought in the modified one, was the changes in the robot co ordinates, however, the edges as well changes to counter these ups. The change in coordinate system in matlab is as shown below:

robot.polygonCoords = gen_pentagon (0.8, -0.30, 0.8, 0.30, -0.19, 0.30, -0.3, 0, -0.19, -0.30);

robot.polygonCoords = gen_pentagon (0.5, -0.30, 0.5, -0.05, 0.75, -0.05, 0.75, -0.3, 0.8, -0.3, 0.8, 0.3, 0.75, 0.3, 0.75, 0.05, 0.5, 0.05, 0.5, 0.30, -0.19, 0.30, -0.25, 0, -0.19, -0.30);

Figure 2.6: Changes in coordinates between the old and new design

```
pentagonm = [ Ax, Ay;
              Bx, By;
              Cx, Cy;
              Dx, Dy;
              Ex,Ey;
              Ax, Ay;];
```

Figure 2.7: Lists of vertexes in the old design

```
pentagonm = [  Ax, Ay;
               Bx, By;
               Cx, Cy;
               Dx, Dy;
               Ex, Ey;
               Fx, Fy;
               Gx, Gy;
               Hx, Hy;
               Ix, Iy;
               Jx, Jy;
               Kx, Ky;
               Lx, Ly;
               Mx, My;
               Ax, Ay;];
```

Figure 2.8: Lists of vertexes in the new design

From the change in coordinate system in the m-file till the lists of vertexes prove that the increase in vertexes is corresponds to the new added coordinates. Since the shape is almost the same, changes to main body frame of the robot has less changes, however the new added coordinates are basically on the new added end effector.

Chapter 3

Forces and Torque

3.1 Introduction to Force Equations

The approach by Liu and Wu shows a pushing box strategy which basically on applying repulsive of forces. They asserted that Hooke's law is necessary in modeling this application. They stated that the group of robots can push the box if and only if their distances to the box are less or equal to the defined radius *Io* [3]. The force for each robot is calculated according the formula given below.

$$F = \begin{cases} n(l_0 - di), & if\ di \le l_0 \\ 0, & otherwise \end{cases} \qquad \textit{Equation 1}$$

di – Euclidean distance
n - Positive coefficient
l_0 - Range of the box

3.2 How this force equation is implemented in MATLAB software

In approaching this method (Behavior approach) asserted by Liu and Wu, the first step is creating the boundary which surrounds the moveable object, range of the box (l_0). This range is established by setting a specific distance between the moveable object and the robot. As soon as the robots enter this region, the robots are able to apply repulsive force on the object and the object will move provided that enough force has been applied. The codes showed in Figure 3.1 displays the idea of how this equation (*Equation 1)* is implemented in matlab.

```
F1 = ((n*(2.3-do1)) + 2); %force calculation for robot 1
F2 = ((n*(2.3-do2)) + 2); %force calculation for robot 2
```

l_0

Positive coefficient (n = 10)

Figure 3.1: Force equation in Matlab

```
do1 = MovObjectPoseCurrent (1) - pose Current (1); % Euler distance
do2 = MovObjectPoseCurrent (1) - pose Current (1); %Euler distance
```

Figure 3.2: Calculation of Euler distance between the object and the robots

To contrast, *Equation 1* is quite different with force equation shown in figure 3.1. However, the overall picture is totally the same. The only factors that differentiated the force equation in

figure 3.1 are the positive coefficients (n) and 2.3 (I_0). The positive coefficient is chosen using the try error method such as the n variable keep tuning or increasing until the object movements seems to responds well to the applied forces. The radius of the box is chosen in such a way that the robots can only applied repulsive force when it is very close to an object, in this case 2.3 is the best range.

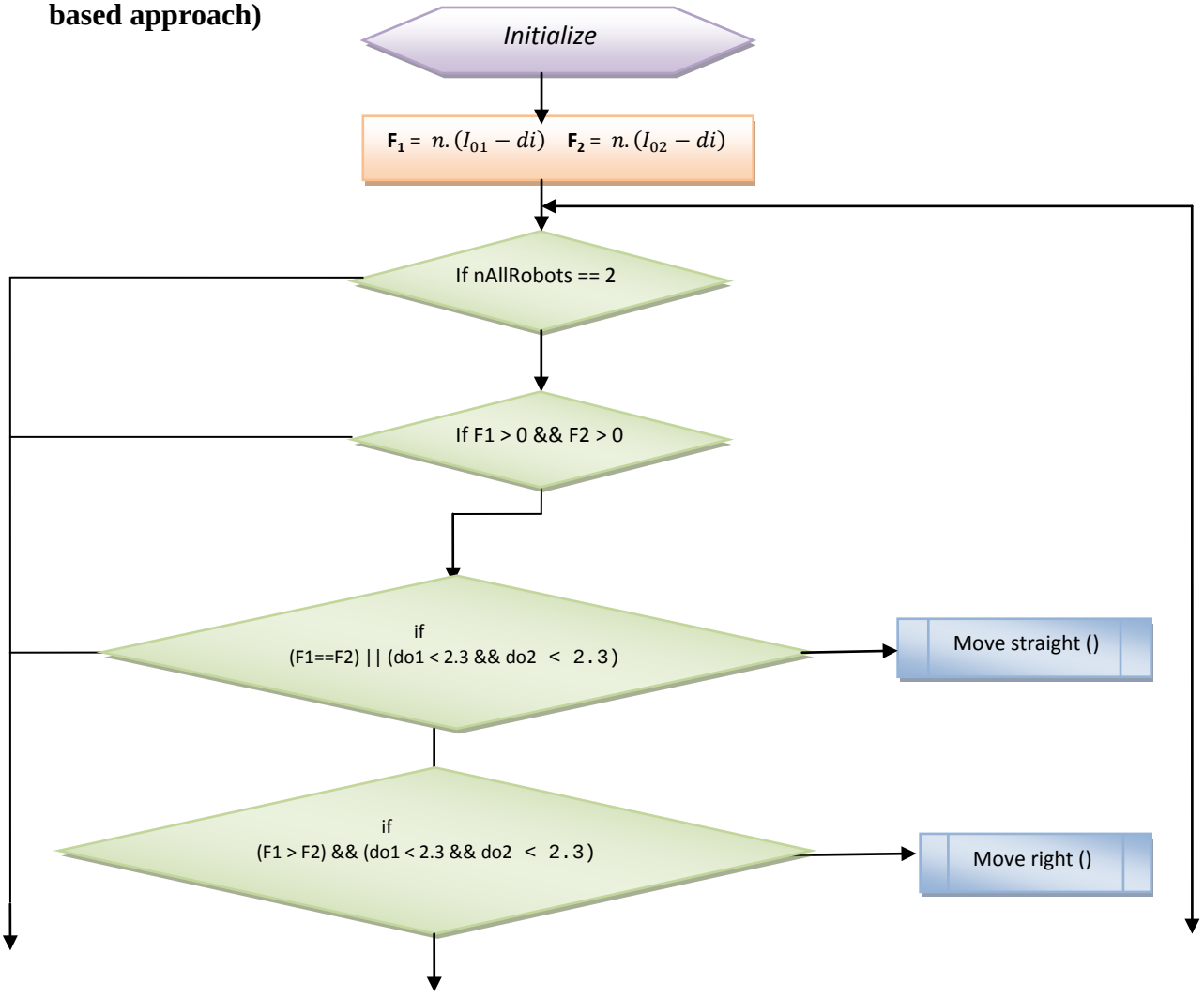
To add, the calculation of Euler distance is shown in figure 3.2. The results show that every movement of the robot contributes the reduction in Euler distance, distance between the robots and an object. While the robots approaching the object, the force is negative since the Euler distance is large (according to *Equation 1*) but while the robots are near an object, force is increasing and become positive.

To take control of the unbalanced movements due to the uneven of forces, error function is implemented to reduce this problem. Error function is originated from Multiple Impedance control approach; however the error function used in this project is simplified to alleviate its complexity. The error function is as shown below:

Err = MovObjectPoseCurrent (3) – MovObjectPoseTarget (3)

Figure 3.3: Err is equal to the different between the object target angle and the robot current angle

3.2.1 Flow chart with controlled force algorithm (Multiple Impedance Law and Behavior based approach)



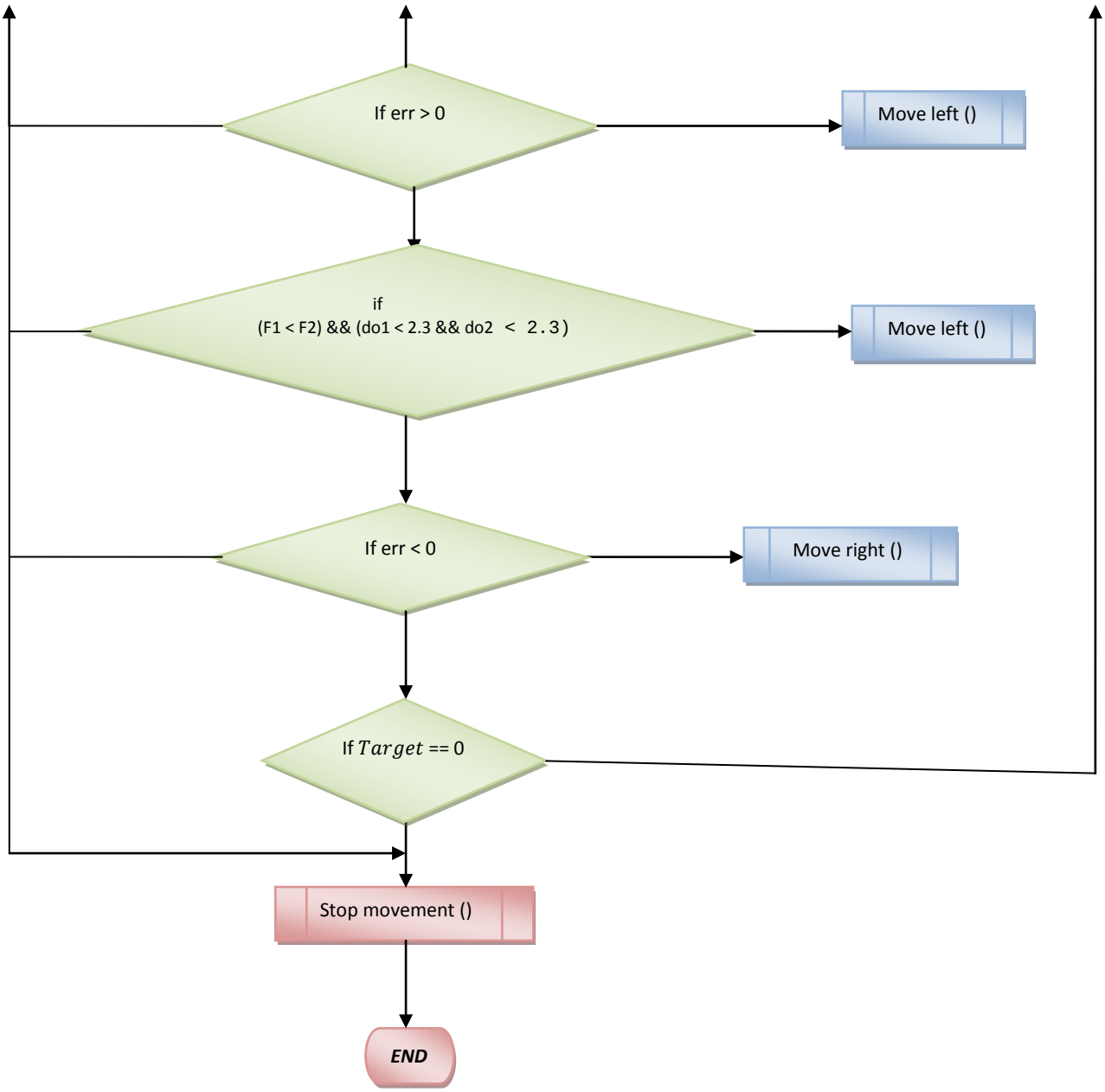


Figure 3.4: Controlled force flow chart using multiple impedance law (MIC) and Behavior based approach

As illustrated from the flow chart shown in **Figure 3.4**, the start off begins with calculating the forces for each robot using *Equation 1*. Next, the number of robot is checked and if one appears, only one robot will do the task however continuing the chain by comparing the forces between the two robots. The comparison takes within an if-else control system. First forces are checked if they greater zero, then it continues to next level however, the object still stationary. In the next phase, the forces are examined and if the two forces equal or the robots are inside the pushing region, the object is moving straight. However if the force of robot one is greater than that of robot two and both are within the region, the object is moving to the right. Then the error function comes about, if Err is greater than zero, since the object has turned due the unbalance force, then robot two is forced to apply greater force to turn the object left as stated from the error function (Err > 0). Similarly, this applies for the opposite condition. If robot two applies more force causing the object to turn left, robot one will be forced to turn the object right and now the error function is less than zero. The error function must keep at minimum value to prevent the unbalanced movement of the object. The iteration keep going and the object stop moving only if the object has reached the target or the robots are off their desired position and outside the range of the box (I_0').

The movement of the object with respect to the three different functions (move right, move left and move straight) are modeled using Newton second law of motion combined with vector composition. The first and second functions (move right and move left) are developed by involving the equation of angular momentum:

$$\sum \boldsymbol{M} = \boldsymbol{I\omega} \qquad \textit{Equation 2}$$

M - *Angular momentum (kg.m^2/s)*
I - *Moment of Inertia (kg.m^2)*
ω - *Rad/s*

Thus, equation 2 shows that the object is moving with an angular velocity (ω) while being pushed on any fixed position to either side of the object (right or left).

The use of vector composition is applied for finding the direction of movement and this implied for finding the resultant force of two robots. The last function (move straight) involves no angular momentum, in fact the object will move straight if and only if the applied forces were same in magnitude and direction.

3.3 Torque

When an object is pushed with a force directed towards its center of mass, the object will accelerate. But it will not start rotating about its center of mass (no torque is applied therefore implies no angular acceleration.)

But when object is pushed with a force not directed towards the center of mass, a torque about the CM is exerted, because the force now has a lever arm. This will result in linear as well as angular acceleration of the object. The linear acceleration is a result of the force and the angular acceleration is a result of the torque.

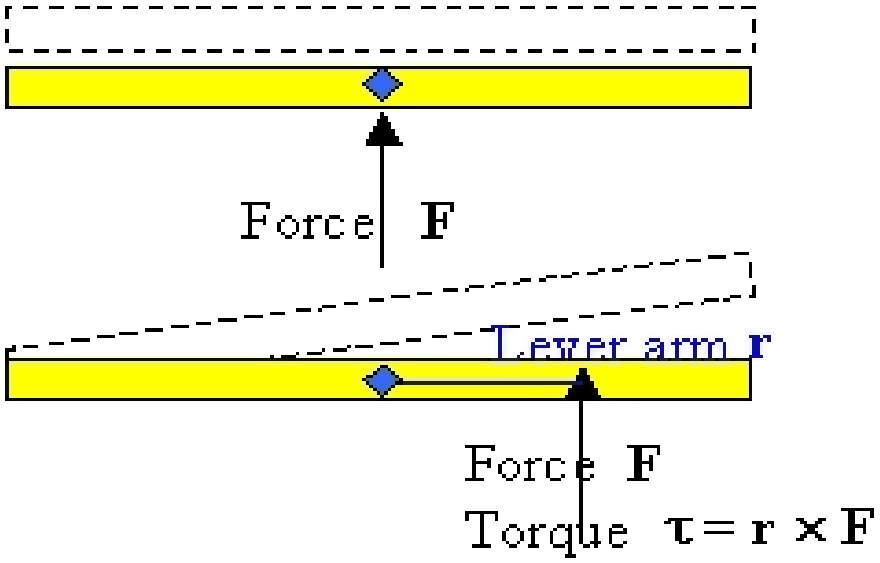


Figure 3.5: Basic Torque Model

3.3.1 Finding torque

Torque can be calculated using Euler's equation for motion

$$\tau = I\alpha + \dot{\omega}.A$$

α-angular acceleration(rad/s^2)

I-polar moment of Inertia (kgm^2)

$\dot{\omega}$-angular velocity(rad/s)

A-coefficient of friction

Chapter 4

Friction

4.1 Overview

An important factor to consider when pushing an object is Friction. The basic definition of friction could be resistance in the opposite direction of motion. Therefore it can be modeled as with a force usually referred to as the frictional force. The figure below shows the basic model of forces on an object on a flat plane.

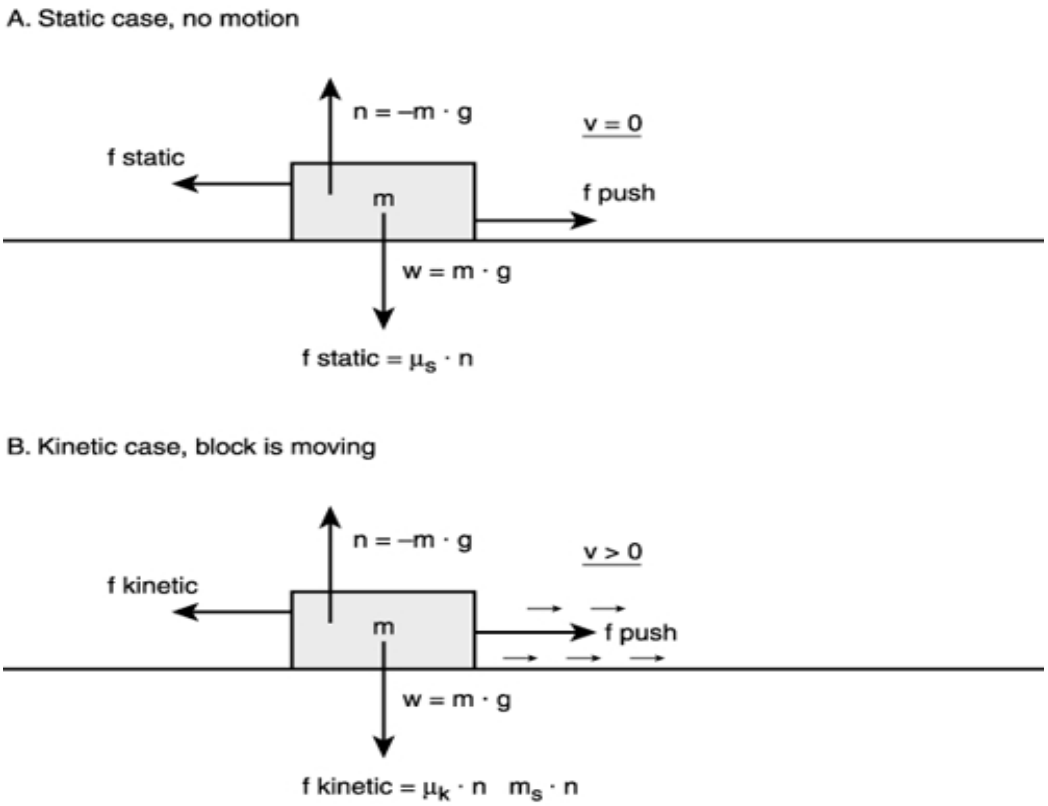


Figure 4.1: Basic model of forces on an object on a flat plane

From the above figure it can be established that there are two types of friction:

- Static Friction

$$F_{fstatic} = m*g*\mu_s.$$

- Kinetic Friction

$$F_{fkinetic} = m*g*\mu_k.$$

M-mass of the object

g- Gravitational constant (9.8 m/s^2)

U_S/U_K – Being the coefficient of friction

In real circumstances when an object is pushed along different types of surfaces there will be different values of coefficient applied. The table below shows the coefficient of U_S and U_K along different surfaces.

Table 4.1: Coefficient of U_S and U_K along different surfaces.

Coefficient of Friction		
Surfaces	**Static Friction**	**Kinetic Friction**
Steel on steel (dry)	0.6	0.4
Steel on steel (greasy)	0.1	0.05
Teflon on steel	0.041	0.04
Brake lining on cast iron	0.4	0.3
Rubber tires on dry pavement	0.9	0.8
Metal on ice	0.022	0.02
Rubber tip of crutch on rough wood	0.7	--

The Standard Friction equation is the relationship between the resistive force of sliding friction for hard surfaces, the normal force and the coefficient of friction between the two surfaces. When applied to sliding friction of hard surfaces, it implies that friction is independent of the area of the surfaces in contact.

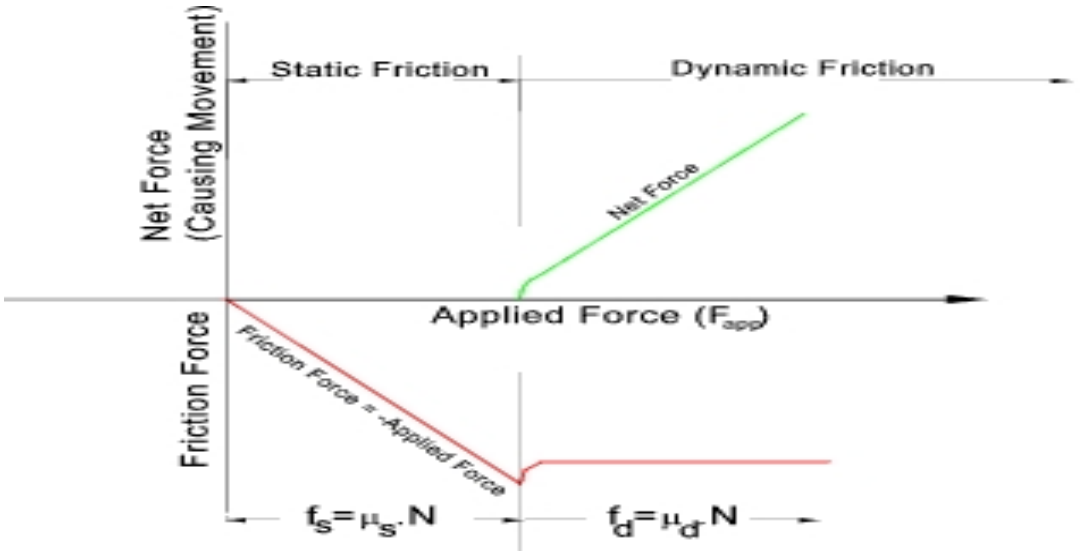


Figure 4.2: Graph of friction force and Net Force

The above **Figure 4.2** shows how the Net force (causing movement) behaves with regards to Friction force and applied force. As seen in the graph in static friction the net force is cancelled out and therefore no movement is possible. Although as soon as enough force is applied to overcome static friction f_s, the object is able to move. However, the opposing force is now dynamic or kinetic friction.

4.2 Model Calculation

Sum of forces in the x-direction

$$\sum_{i=1}^{n} Fi = F_n - F_{\mu k}$$

F_N – Resultant force (R) on (n) forces, $F_{\mu k}$ - Static friction force

Since $F_1 = F_2 = 3N$, therefore $F_N = F_1 + F_2 = 6N$

Calculating the net force

$$F_{\mu k} = \mu_k . N$$

$$F_{\mu k} = 0.3 \; x \; 4N = \mathbf{1.2N}$$

$$F_{net} = F_N - F_{\mu k}$$

$$F_{net} = 6N - 1.2N = 4.8N$$

$$a_{net} = {}^{4.8\ N}/_{0.5\ kg + 0.4\ kg} = 5.33\ m/s^2$$

Chapter 5

Experiments, Results and Discussion

5.1 Experimentation Methodology

The experiment was carried by compiling three setups (each setup consists of ten trials) for pushing an object by the group of robots for a short run (3 meters). Then the simulations extended for longer distances. The first experiment based on a simple scenario, the robots have to push the object for only three meters. For experiment two, the same strategy used however this considers the movements of the robots. In experiment three, the robots continue pushing the object but after 9.1 meters, robot one turned off and left for robot two to push an object. After 10.1 meters elapsed, robot one activated and with robot two already online the two continued together to complete task. After the three important tests, continue simulations were recorded. These simulations were focus on testing the performance of the group of robots in pushing the object for longer distances (greater than 3 meters).

Table 5.1.1: Experiment to consider the movement of the object within 3 meters

Trials	**Status (F/S)**	**Time Elapsed (s)**	**Theta**	**Distance from Target**
1	S	17.44	0.04	-0.03
2	S	17.18	0	0.02
3	S	17.33	0	-0.03
4	S	16.58	0	-0.02
5	S	16.55	0	-0.02
6	S	16.79	0.01	-0.04
7	S	16.5	0.01	-0.01
8	S	16.78	0.04	-0.01
9	S	16.74	0	-0.02
10	S	16.94	0.01	-0.01
	Mean	**16.88**	**0.01**	**-0.02**

Table 5.1.2: An experiment to consider the movement of the robots within 3 meters

Trials	Status (F/S)	Time Elapsed (s)	Theta	Robot 1 (target)	Robot 2 (target)	Desired	Current 1	Current 2
1	S	19.52	0	0.04	0.03	2.35	2.31	2.32
2	S	19.04	0.07	-0.01	0.04	2.35	2.36	2.31
3	F	19.57	0	0.06	0.03	2.35	2.29	2.32
4	S	19.81	0.1	0.01	0.04	2.35	2.34	2.31
5	F	19.54	0.01	0.06	0.04	2.35	2.29	2.31
6	S	19.8	0	0.04	0.03	2.35	2.31	2.32
7	F	19.93	0	0.05	-0.01	2.35	2.3	2.36
8	F	19.88	0.01	0.06	0.08	2.35	2.29	2.27
9	S	19.66	0.01	0.05	0	2.35	2.3	2.35

10	S	19.91	0.01	0.04	0.03	2.35	2.31	2.32
	Mean	19.67	0.02	0.04	0.03	2.35	2.31	2.32

Table 5.1.3: An experiment to test the performance of the robots under failure conditions

Trials	Status (F/S)	Time Elapsed (s)	Theta	Distance from Target
1	S	21.71	-0.07	-0.03
2	F	21.73	-0.34	0.99
3	F	22.73	0	-0.03
4	S	22.78	-0.13	-0.03
5	F	22.32	-0.34	0.96
6	S	22.37	-0.09	-0.03
7	F	32.7	-0.24	2.28
8	F	24.95	-0.34	0.54
9	S	24.25	-0.64	-0.03
10	F	25.21	-0.5	0.45
	Mean	**24.08**	**-0.27**	**0.51**

5.2 Experiments for testing the performance of the robots in pushing an object for longer distances

5.2.1 Simulation snapshots

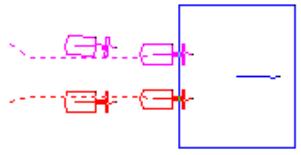


Figure 5.2.1: Two robots approaching an object

Figure 5.2.2: One robot is selected to do the task

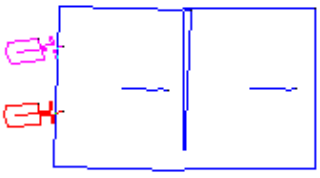


Figure 5.3: The object is a bit off the desired trajectory

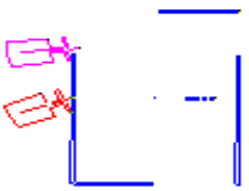


Figure 5.4: An object reaching the target

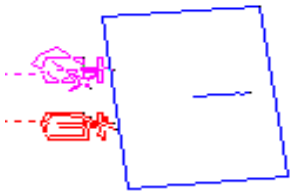


Figure 5.5: Uneven force applied by the bottom robot

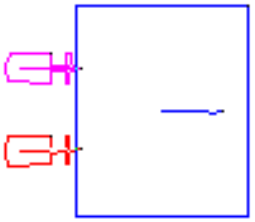


Figure 5.6: Even force applied by the robots

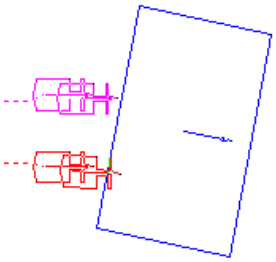


Figure 5.7: The bottom robot is pushing the object

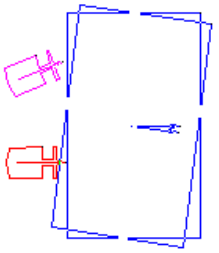


Figure 5.8: Uneven force applied at the target

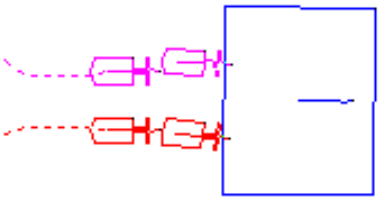


Figure 5.9: Uneven forces applied by the two robots

5.3 Velocity Graph

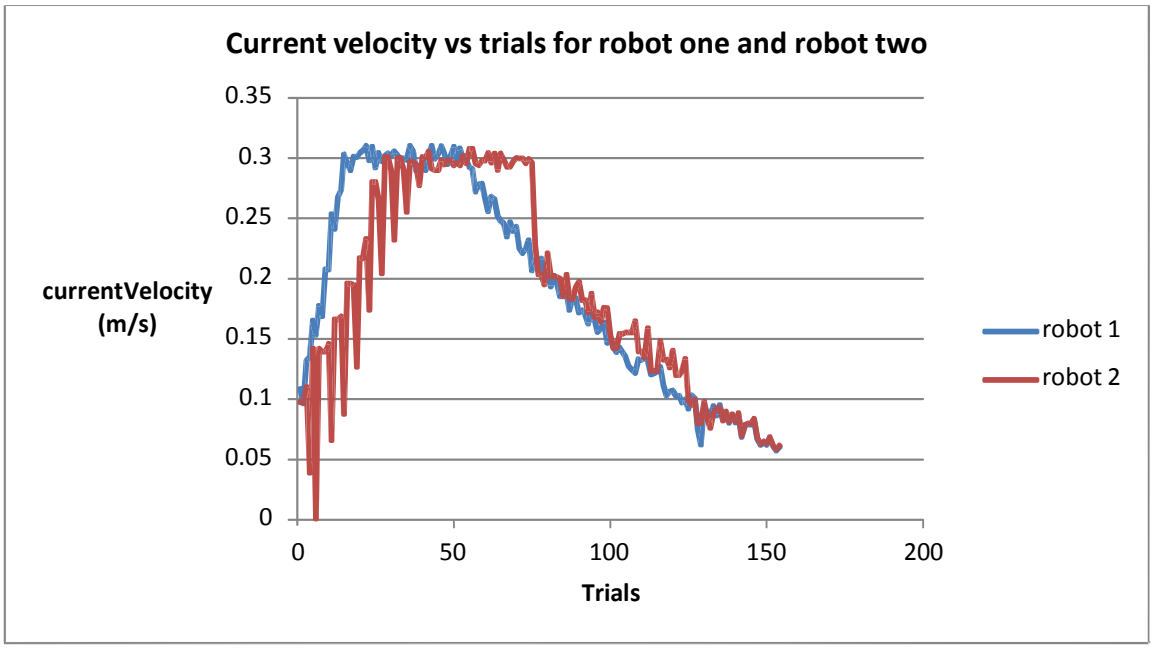


Figure 5.3 Graph of current velocities of two robots approaching an object

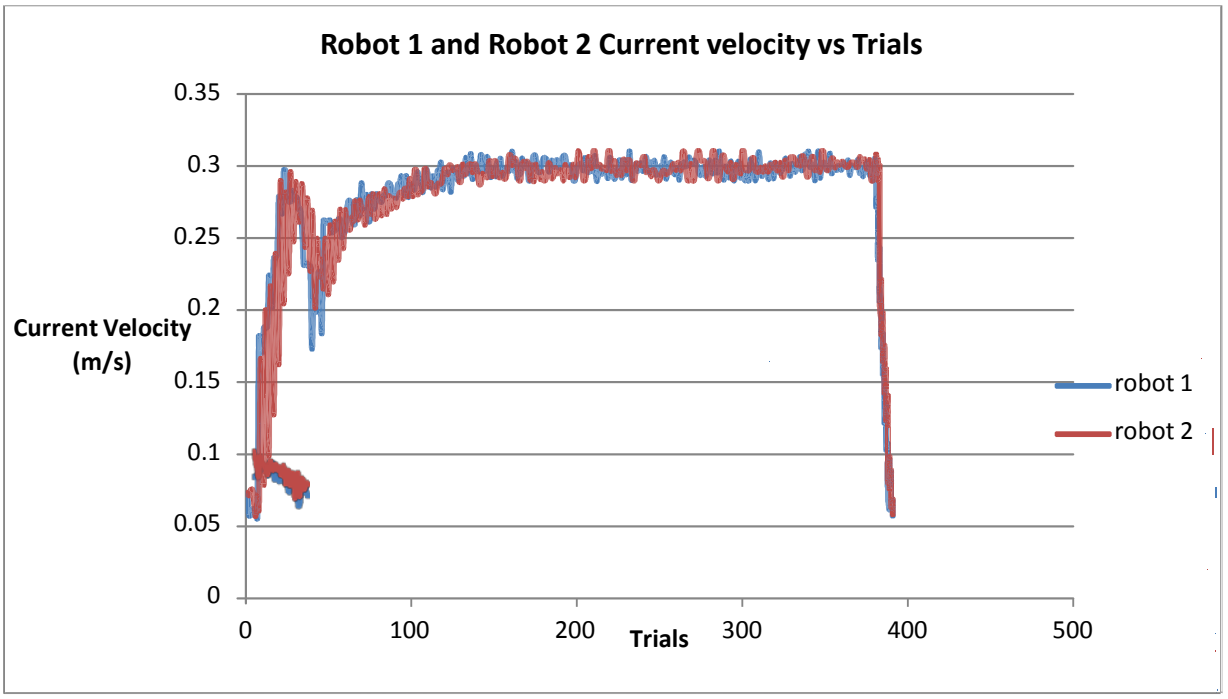


Figure 5.4 Graph of current velocities of the two robots while pushing an object to the target location

5.4 Force graph

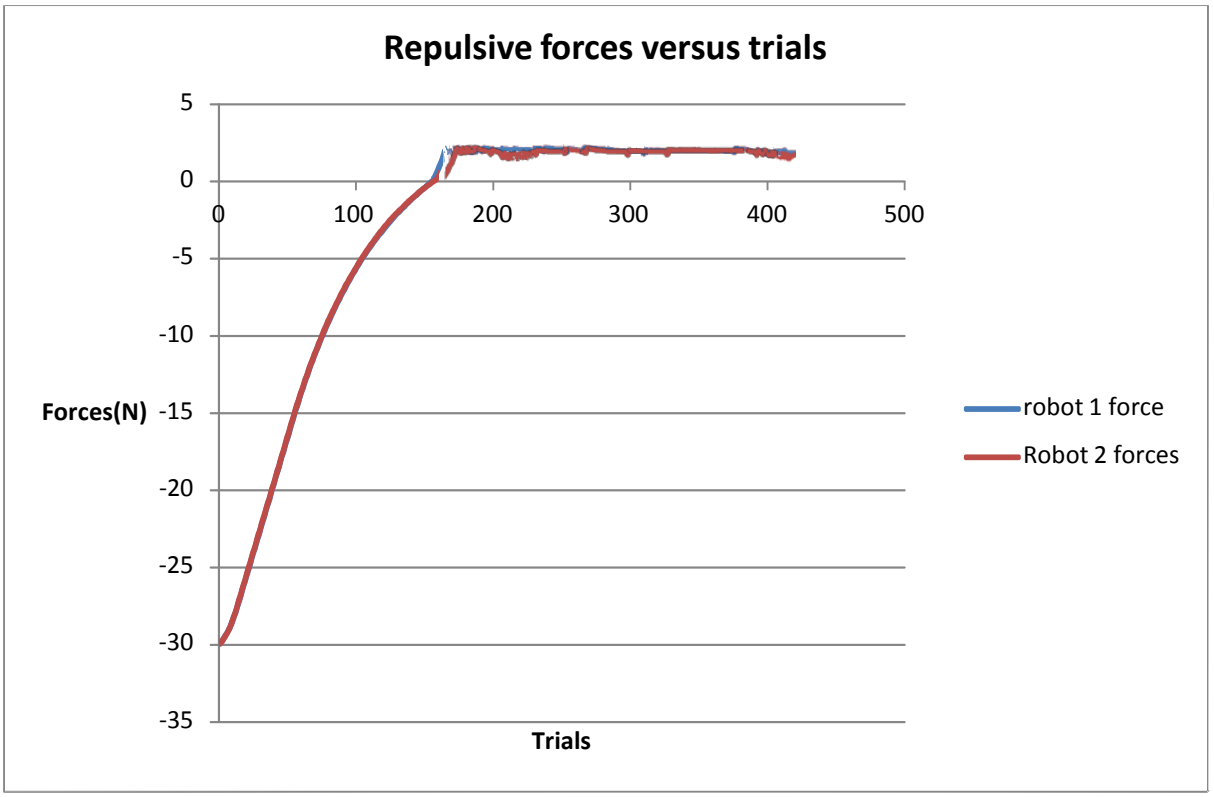


Figure 5.5 Graph of Repulsive force graph for robot 1 and robot 2

5.5 Discussion

The first runs in the experiments are basically on testing the performance of the group of robots in pushing an object. In this first experiment, two robots had to push that object for short distance (3 meters from the starting line of the object). The status (F/S) represents fail and success, labeled success if the object close to the target location by ± 0.5m otherwise fail (F). From the results in **Table 5.1.1**, no error recorded for the movement of the object to the goal line and the mean time for this task to complete was 16.88 seconds. The next experiment was based on the same scenario, except the movements of the robots are measured and compared with their target values. To add, **Table 5.1.2** shows that more error now taking place in the third, fifth and the eighth trials since one of the robots are off by ± 0.5 meters from their desired positions. In the last experiment for short distance run, error overwhelmed the results. The experiments proved that the robots were unable to recover if one robot fails at the middle of the operation. The mean time for this operation was recorded as 24.08 seconds.

The next experiment in section 5.2 started off with task allocation strategy for the initial selection of robots. In this implementation, two types of robots were standby for the call. The call upon different robots depended on type of the moveable objects. Each moveable object was allocated with a different identification number (ID). For example; if the ID is equal to one two robots were sent to execute the task otherwise a single robot is called to execute the

task. In the first iteration, two robots were sent to execute the task since the ID number of the moveable object was one. The first target location was calculated in such a way that the robots approached a single point of contact from the lists of contact points around the object edges, as shown in **figure 5.2.1**. Furthermore, the change in velocity while the robots approaching the object was shown in **Section 5.3**. The graph clearly showed that the two robots initially increased their current velocities however the velocity cut down after fifty runs. The reason for the reductions in velocity signifies that the robots were near to their first target location.

Furthermore, selected simulations were chosen from the lists of simulation snapshots to give an overview how the two robots managed to push the object to the target location. In **Figures 5.2.5, 5.2.7** and **5.2.8**, the object experienced an unbalanced forces applied by the two robots which dragged the object off the desired trajectory. Similarly, in **figure 5.2.8** the object was pushed after the object has reached the target location and this may results from the lack of controlling the pushing forces applied by the two robots.

To take control of the contacts forces between the moveable object with the end contacts of the robots, two methods were needed to implement. The first methodology was Multiple Impedance law (MIC) while the second was Behavior Based Control (BBC). MIC was based on controlling the end - effectors current positions with the current positions of the moveable object. In this project, these two approaches were combined and important concepts applied were similar to the basics concept of each approach. However most of the parts applied in this project were based on BBC and the force calculations were discussed in detailed in chapter 3. Apart from this, error function was used and this originated from MIC. The error function calculates the difference between the target angle and current angle of the object. As the error function was greater or less than zero, one of the robots was expected to apply greater force than its peer in order to turn the object to its expected route. From the lists of simulation results, **figure 5.2.5** and **figure 5.2.7** illustrated these cases. In figure 4.5, uneven forces were applied by the two robots which cause the imbalance in the object movements. These unbalanced movements caused the fluctuating of error function. For simplicity as the error went below zero, robot one was forced to apply more force than robot two otherwise robot two would apply more force. Similarly, **figure 5.2.7** displayed the same case however robot two compensated the unbalanced force and try to drive the object to the desired trajectory instead of robot one. However, even force applied by the two caused less unbalanced movements of the object and this proved in **figure 5.2.6**.

The magnitude of velocities shown in **Figure 5.4** demonstrated the responds of the two robots while pushing the object. The first hundreds runs of the two robots were inconsistent but can be said that the two robots were increasing their current velocities and this cause to overcome the friction force the object stored. The next runs showed the fact that the object was moving since the robots was travelling in constant velocities, however fluctuating in velocities signifies the uneven of forces being applied while pushing the robots to keep the object travelling on the desired trajectory. The cut down region showed that the object has reached its location and the robots' velocities were forced to reduce to avoid further pushing of the object.

The next chart shown in **section 5.4**, given the force trend for robot one and robot two, detailed calculations of force was discussed in chapter 3. The graph displayed that the robots' forces where initially less than zero and exponentially increasing until no further increasing was required. The negative forces came from the fact that the object and robots were separated by great distance (Euler distance). However, as the robots enters the region which radius established for pushing purposes, the force becomes positive and the object started to move in respect to the directions and magnitude of the force. The constant forces displayed from the graph showed the fact that same forces has been applied on the object by the robots in order to keep the object moving in the direction of the resultant force between the two robots.

5.5.1 Problems encountered and expected solutions

As the project concern, there was still one critical part needed to improve in the project. This most important part was force control, the area that needed to be improved since the object was still moving off the target in some iteration, even though force control has implemented. The expected solution was to implement the full concept of MIC as explained below:

As the object starts to move, MIC is taking over BBA to control the force interactions and to control the movement of the object based on the applied force. The force for each robot is based on the concept of the Impedance control [1]. To add, the two major components of MIC and these were used in this project:

1. Position control strategies – this happens when an end effector position (x_2) or the object position (x_3) are provided with better tracking path to the goal location.
2. Force Regulation control – the force of end-effector is controlled based on the acceptable trajectory of the object and presuming that the object is a rigid body. In this case, x_2 is not controlled, F_e could provide tracking errors of x_3:

$$F_e = Ł_2(\dot{x}_2 - \dot{x}_3) + k_2(x_2 - x_3)$$

$Ł_2$ - damping coefficient (N. s^2/ m)

k_2 - Stiffness coefficient (N/m)

Considering the equation of the moving object:

$$m_3\ddot{x}_3 = f_0(x_3, \dot{x}_3) + F_e + F_c$$

m_3 - Mass of the object

f_0 - All potential frictional forces

F_c – contact forces acting on the object

Therefore the approximate controlled force and the error function are as shown below:

$$F_{edes} = m_3\ddot{x}_{3des} - f_c(x_{3des}, \dot{x}_{3des})$$

$$\epsilon_f = m_3(\ddot{x}_{3des} - \ddot{x}_3) - f_0(x_{3des}, \dot{x}_{3des}) + f_0(x_3, \dot{x}_3) + f_c$$

To provide a better controlled force on the object ϵ_f *must be equal to zero*

From the concept shown, it could be concluded that error function used in the project needed to be modified and force contact points between the robots and the object has to be implemented precisely. This could counter the downs of off course situations where the object travelled a different trajectory apart from the expected one.

Chapter 7

Conclusion and Recommendations

7. 1 Conclusions

To conclude, the project has been implemented in Matlab of a multi robot system which consists of robots that push and a particular object across a surface to a predefined target location. The number of robots to execute the task depends on the mass of the object. The robots approach an object by considering a single point from the lists of points around an object. The force equation implemented in this project was based in behaviour based approach and multiple impedance control. The behaviour based approach was used for calculating the force with based on Euler equation of motion. However, multiple impedance control was used to control the contact force between the robots and an object. The basic concepts of multiple impedance control used to control contact force between the object and the robots was mainly on error function. There are different experiments being recorded and showed that the group of robots can operate the task successfully for short and longer distances. However, the experiments also prove that the group of robots were not fit to operate the task under failure conditions since there is no communications established between the robots.

7.2 Recommendations

It can be recommended to those who are undertaking this same project to consider some important areas in order to achieve a better and more effective approach.

- Force equation - There is a need to improve force equations to consider the full control of the object movements.
- Torque – Torque is another area to be improved in this project, thus mechanical torque has been implemented in the project but considering electrical torque can be more effective.
- Understanding MATLAB SOFTWARE – Understanding MATLAB software can be too costly to the project, therefore it is very important to start off the phase with understanding MATLAB software.
- Information – Information about this project through online sources might inadequate therefore researches must consider other sources.

References

[1] Wang.Y.N.Y, "Assessment of team Q-Learning in a cooperative multi-robot task", viewed 25 August 2011 www.cancam2011.dal.ca

[2] Liu.J and Wu. J, 2001, *Multi-Agent Robotic Systems*, New York, United of America

[3] Mataric.M.J atal, "Cooperative Multi robot Box Pushing", viewed 25 August 2011 www.robotics.usc.edu

[4] Ahmadabadi.M.N and Nakano. E, "A "Constrain and Move" Approach to Distributed Object Manipulation", *IEEE Transactions on Robotics and Automation*, vol.4, no.2, pp. 2-3

[5] Alipour.K atal, "Formation Control and Obstacle Avoidance of cooperative Wheeled Mobile Robots", viewed on 25 August 2011, www.aas.net.cn

[6] Study Physics, "Friction", viewed 25 August 2011 www.studyphysics.ca

[7] Moosavian.A.S atal, "Cooperative Object manipulation with contact Impacts Using Impedance Control", viewed 25 August 2011 www.springer.com

[8] Floreano.D and Mattiuss.C, 1996, "Behaviour –based Robots", viewed on 25 August 2011, www.baibook.epfl

[9] Mali.A.D and Mukerjee.A,"Metrics for evaluation of behavior based robotic system" , viewed on 25 August 2011, www.cs.umn.edu

[10] Gini.M and Shackleton.J, 1997, "Measuring the effectiveness of reinforcement learning for behavior based robot", viewed on 25 August 2011, www.cs.umn.edu

Appendices

Appendices

Function for fetching the characteristics of the moveable object (MovObject)

```
function MovObject = get_MovObject_parameters(allMovObjectIDs,iMovObject,~)

MovObjectID = allMovObjectIDs(iMovObject);%

MovObject.MovObjectID = MovObjectID; % additional input - MovObjects
selected from task devolution

switch(MovObjectID) % change to MovObjectID

    case {1,4}


        MovObject.shape = 'non-circular'; % circular or non-circular
        MovObject.velLinMaxPhysical = 1;
        MovObject.radius = 0.5; % radius used by control algorithms
(includes a small safety margin for circular MovObjects 0.45)

        %if (strcmp(MovObject.shape,'non-circular'))
        % specify min and max radius
        MovObject.radiusActual = [0.5 1];
        %end;
%         MovObject.polygonCoords = gen_rectangle(0.45, -0.3, 0.45, 0.3, -
0.19, -0.19, -0.30);
        [MovObject.MovObjectVertex,MovObject.MovObjectEdge] = getcoord();
         MovObject.polygonCoords = MovObject.MovObjectVertex;
        % safety margin approximated by a circle or rectangle


    case {2,5}
                MovObject.shape = 'non-circular'; % circular or non-circular

        MovObject.radius = 2; % radius used by control algorithms (includes
a small safety margin for circular MovObjects 0.45)

        %if (strcmp(MovObject.shape,'non-circular'))
        % specify min and max radius
        MovObject.radiusActual = [0.35 1];
        %end;
%         MovObject.polygonCoords = gen_rectangle(0.45, -0.3, 0.45, 0.3, -
0.19, -0.19, -0.30);
        [MovObject.MovObjectVertex,MovObject.MovObjectEdge] = getcoord();
        MovObject.polygonCoords = MovObject.MovObjectVertex;
        % safety margin approximated by a circle or rectangle


    case {3,6}
                MovObject.shape = 'non-circular'; % circular or non-circular

        MovObject.radius = 2; % radius used by control algorithms (includes
a small safety margin for circular MovObjects 0.45)

        %if (strcmp(MovObject.shape,'non-circular'))
        % specify min and max radius
        MovObject.radiusActual = [0.35 1];
        %end;
```

```
%         MovObject.polygonCoords = gen_rectangle(0.45, -0.3, 0.45, 0.3, -
0.19, -0.19, -0.30);
        [MovObject.MovObjectVertex,MovObject.MovObjectEdge] = getcoord();
        MovObject.polygonCoords = MovObject.MovObjectVertex;
        % safety margin approximated by a circle or
rectangle_achievement_rates(MovObjectID);

    otherwise
        message = ['Reference to non-existent parameters for MovObject'
num2str(MovObjectID) '!'];
        error(message);
end;



%exchange map data flag
MovObject.isMapExchanged = 0;

MovObject.poseCurrent(1) = 8.5;
MovObject.poseTarget(1) = 15.6;
MovObject.poseCurrent(2) = 5.5;% + (MovObjectID - 1);
MovObject.poseTarget(2) = 5.5;% + (MovObjectID - 1);
MovObject.poseCurrent(3) = 0; %pi
MovObject.poseTarget(3) = 0;
```

Function for drawing the layout of the MovObject

```
function [MovObjectVertex,MovObjectEdge] = getcoord()

MovObjectVertex = [ 1.4, -2.2, 0;%1(10,7)-(1, -1)
                    1.4, 2.2, 0;%2(8,7)-(1,  1)
                   -1.4, 2.2, 0;%1.4(8,4)-(-1,  1)
                   -1.4, -2.2,0;%4(10,4)-(-1,  -1)
                    1.4, -2.2, 0];%1(10,7)-(1,  -1)

MovObjectEdge = [1,2;
                 2,3;
                 3,4;
                 4,1];
```

Function for making the MovObject turning (Moving left)

```
function [MovObjectVertex,MovObjectPoseCurrent]=
Move_left(MovObjectPoseCurrent,MovObjectVertex)


[MovObjectPoseCurrent(1),MovObjectPoseCurrent(2)] =
position_add(MovObjectPoseCurrent(1),MovObjectPoseCurrent(2),0.04,0.00);%0.0
001
MovObjectPoseCurrent(3) = MovObjectPoseCurrent(3) - 0.01;
```

Function for making the MovObject turning (Moving right)

```
function [MovObjectVertex,MovObjectPoseCurrent] =

  Move_left(MovObjectPoseCurrent,MovObjectVertex)


[MovObjectPoseCurrent(1),MovObjectPoseCurrent(2)] =

position_add(MovObjectPoseCurrent(1),MovObjectPoseCurrent(2),0.04,0.00);
```

```
MovObjectPoseCurrent(3) = MovObjectPoseCurrent(3) + 0.01;
```

Function for adjusting the target location of the robots when approaching the object

```
if(nAllRobots == 2)
[Rob1TargetY,Rob1TargetX,Rob2TargetY,Rob2TargetX] =
getNewTarget(globalEnvData.MovObjectVertex,MovObject{allMovObjectIDs(iMovObj
ect)}.poseCurrent);

        robot{allRobotIDs(1,1)}.poseCurrent(1) = 3;
        robot{allRobotIDs(1,1)}.poseTarget(1) = Rob1TargetX;
        robot{allRobotIDs(1,1)}.poseCurrent(2) = 4.5;% +
(robot{allRobotIDs(1,1)}ID - 1);
        robot{allRobotIDs(1,1)}.poseTarget(2) = Rob1TargetY;% +
(robot{allRobotIDs(1,1)}ID - 1);
        robot{allRobotIDs(1,1)}.poseCurrent(3) = 0; %pi
        robot{allRobotIDs(1,1)}.poseTarget(3) = 0;

        robot{allRobotIDs(2,1)}.poseCurrent(1) = 3;
        robot{allRobotIDs(2,1)}.poseTarget(1) = Rob2TargetX;
        robot{allRobotIDs(2,1)}.poseCurrent(2) = 6.3;% +
(robot{allRobotIDs(2,1)}ID - 1);
        robot{allRobotIDs(2,1)}.poseTarget(2) = Rob2TargetY;% +
(robot{allRobotIDs(2,1)}ID - 1);
        robot{allRobotIDs(2,1)}.poseCurrent(3) = 0; %pi
        robot{allRobotIDs(2,1)}.poseTarget(3) = 0;

else
    [Rob1TargetY,Rob1TargetX] =
getNewTarget1(globalEnvData.MovObjectVertex,MovObject{allMovObjectIDs(iMovOb
ject)}.poseCurrent);

        robot{allRobotIDs(1,1)}.poseCurrent(1) = 3;
        robot{allRobotIDs(1,1)}.poseTarget(1) = Rob1TargetX;
        robot{allRobotIDs(1,1)}.poseCurrent(2) = 4.5;% +
(robot{allRobotIDs(1,1)}ID - 1);
        robot{allRobotIDs(1,1)}.poseTarget(2) = Rob1TargetY;% +
(robot{allRobotIDs(1,1)}ID - 1);
        robot{allRobotIDs(1,1)}.poseCurrent(3) = 0; %pi
        robot{allRobotIDs(1,1)}.poseTarget(3) = 0;
end;
```

```
function [Rob1TargetY,Rob1TargetX,Rob2TargetY,Rob2TargetX] =
getNewTarget(MovObjectVertex,poseCurrent)

% newSavVertex = [];
newSavVertex(1,:) = MovObjectVertex(1,:) + poseCurrent;
newSavVertex(2,:) = MovObjectVertex(2,:) + poseCurrent;
newSavVertex(3,:) = MovObjectVertex(3,:) + poseCurrent;
newSavVertex(4,:) = MovObjectVertex(4,:) + poseCurrent;
newSavVertex(5,:) = MovObjectVertex(5,:) + poseCurrent;

% newSavVertex
Rob1TargetY = newSavVertex(4,2) + 1.5;
Rob1TargetX = newSavVertex(4,1) - 0.7;

Rob2TargetY = newSavVertex(3,2) - 1.5;
Rob2TargetX = newSavVertex(4,1) - 0.7;
```

```
function [Rob1TargetY,Rob1TargetX] =
getNewTarget1(MovObjectVertex,poseCurrent)

% newSavVertex = [];
newSavVertex(1,:) = MovObjectVertex(1,:) + poseCurrent;
newSavVertex(2,:) = MovObjectVertex(2,:) + poseCurrent;
newSavVertex(3,:) = MovObjectVertex(3,:) + poseCurrent;
newSavVertex(4,:) = MovObjectVertex(4,:) + poseCurrent;
newSavVertex(5,:) = MovObjectVertex(5,:) + poseCurrent;

% newSavVertex
Rob1TargetY = newSavVertex(4,2) + 1;
Rob1TargetX = newSavVertex(4,1) - 0.7;

% Rob2TargetY = newSavVertex(3,2) - 1.5;
% Rob2TargetX = newSavVertex(4,1) - 0.7;
```

Experiment 5.1 (Short run)

```
do1 = globalEnvData.MovObjectPoseCurrent(1) -
robot{allRobotIDs(1,1)}.poseCurrent(1);
do2 = globalEnvData.MovObjectPoseCurrent(1) -
robot{allRobotIDs(2,1)}.poseCurrent(1);
        n = 10;

 F1 = ((n*(2.3-do1)) + 2);
 F2 = ((n*(2.3-do2)) + 2);
%  F1
%  F2
 err = (globalEnvData.MovObjectPoseCurrent(3) -
globalEnvData.MovObjectPoseTarget(3));
%  globalEnvData.MovObjectPoseCurrent(1)
if(globalEnvData.MovObjectPoseCurrent(1) > 9.1 &&
globalEnvData.MovObjectPoseCurrent(1) < 9.3)
    robot{allRobotIDs(1,1)}.isExecuting = 0;
    robot{allRobotIDs(2,1)}.isExecuting = 1;
else
    robot{allRobotIDs(1,1)}.isExecuting = 1;
    robot{allRobotIDs(2,1)}.isExecuting = 1;
end;

if(nAllRobots == 2)
    if((F1 > 0 && F2 > 0) || F2>F1)
        if(F1==F2 || (do1 < 2.3 && do2 < 2.3))

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
forceformation(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvD
ata.MovObjectVertex,globalEnvData.MovObjectPoseTarget);
               for iMovObject = 1:nAllMovObjects
                 % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                 % Update target pose.
                 MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
               end;
        elseif( F1 > F2 || (do1 < 2.3 && do2 > 2.3))
```

```
[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                  % Update target pose.
                  MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
                if(err > 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_left(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.M
ovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                  for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                  % Update target pose.
                     MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                  end;
%                   robot{allRobotIDs(1,1)}.poseCurrent(3) =
robot{allRobotIDs(1,1)}.poseCurrent(3) + 0.001;
                  robot{allRobotIDs(1,1)}.poseCurrent(1) =
robot{allRobotIDs(1,1)}.poseCurrent(1) + 0.01;
%                   robot{allRobotIDs(2,1)}.poseCurrent(3) =
robot{allRobotIDs(2,1)}.poseCurrent(3) - 0.001;
                  robot{allRobotIDs(2,1)}.poseCurrent(1) =
robot{allRobotIDs(2,1)}.poseCurrent(1) + 0.01;
                else

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                  for iMovObject = 1:nAllMovObjects
                     % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                     % Update target pose.
```

```
                    MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                  end;
                end;

        elseif(F1 < F2 && do2 < 1.8 &&
globalEnvData.MovObjectPoseCurrent(3)<-0.6 )%|| (do1 > 2.3 && ))

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                  % Update target pose.
                  MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
              if(robot{allRobotIDs(1,1)}.isExecuting == 1 && do1 < 2.4)
                 if(err < 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_left(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.M
ovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                  for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                  % Update target pose.
                     MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                  end;
%                    robot{allRobotIDs(1,1)}.poseCurrent(3) =
robot{allRobotIDs(1,1)}.poseCurrent(3)+ 0.001;
                  robot{allRobotIDs(1,1)}.poseCurrent(1) =
robot{allRobotIDs(1,1)}.poseCurrent(1) + 0.01;
%                    robot{allRobotIDs(2,1)}.poseCurrent(3) =
robot{allRobotIDs(2,1)}.poseCurrent(3) + 0.001;
```

```
                    robot{allRobotIDs(2,1)}.poseCurrent(1) =
robot{allRobotIDs(2,1)}.poseCurrent(1) + 0.01;
                else

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                  for iMovObject = 1:nAllMovObjects
                     % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                     % Update target pose.
                     MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                  end;
               end;
            end;
         end;
     else
         [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
     end;
else
        if(F1 > 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
forceformation(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvD
ata.MovObjectVertex,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                  % Update target pose.
                  MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
        else
           [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent]
=
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
        end;
end;
globalEnvData.MovObjectPoseCurrent(1)
globalEnvData.MovObjectPoseCurrent(3)
```

```
% [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
%  robot{allRobotIDs(1,1)}.velLinMaxPhysical
if(globalEnvData.MovObjectPoseCurrent(1) >
(globalEnvData.MovObjectPoseTarget(1) - 4))

     robot{allRobotIDs(1,1)}.velCurrent = robot{allRobotIDs(1,1)}.velCurrent
- 0.05;
     robot{allRobotIDs(2,1)}.velCurrent = robot{allRobotIDs(2,1)}.velCurrent
- 0.05;
end;
if(globalEnvData.MovObjectPoseCurrent(1) >
(globalEnvData.MovObjectPoseTarget(1) - 1))

     robot{allRobotIDs(1,1)}.velCurrent = robot{allRobotIDs(1,1)}.velCurrent
- 0.06;
     robot{allRobotIDs(2,1)}.velCurrent = robot{allRobotIDs(2,1)}.velCurrent
- 0.06;
end;
if(globalEnvData.MovObjectPoseCurrent(1) >
(globalEnvData.MovObjectPoseTarget(1) - 0.5))

     robot{allRobotIDs(1,1)}.velCurrent = robot{allRobotIDs(1,1)}.velCurrent
- 0.04;
     robot{allRobotIDs(2,1)}.velCurrent = robot{allRobotIDs(2,1)}.velCurrent
- 0.04;
end;
if(globalEnvData.MovObjectPoseCurrent(1) >
globalEnvData.MovObjectPoseTarget(1))
 robot{allRobotIDs(1,1)}.isExecuting = 0;
 robot{allRobotIDs(2,1)}.isExecuting = 0;
 [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
end;
```

Experiment 5.2 (Long run)

```
do1 = globalEnvData.MovObjectPoseCurrent(1) -
robot{allRobotIDs(1,1)}.poseCurrent(1);%Euler distance
do2 = globalEnvData.MovObjectPoseCurrent(1) -
robot{allRobotIDs(2,1)}.poseCurrent(1);
        n = 10;

 F1 = ((n*(2.3-do1)) + 2);
 F2 = ((n*(2.3-do2)) + 2);

err = (globalEnvData.MovObjectPoseCurrent(3) -
globalEnvData.MovObjectPoseTarget(3));

if(nAllRobots == 2)
    if(F1 > 0 && F2 > 0)
        if(F1==F2 || (do1 < 2.3 && do2 < 2.3))

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
forceformation(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvD
ata.MovObjectVertex,globalEnvData.MovObjectPoseTarget);
               for iMovObject = 1:nAllMovObjects
                 % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                 % Update target pose.
```

```
                MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
              end;
        elseif( F1 > F2 && (do1 < 2.3 && do2 > 2.3))

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
              for iMovObject = 1:nAllMovObjects
                % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                % Update target pose.
                MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
              end;
              if(err > 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_left(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.M
ovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                % Update target pose.
                   MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
%                 robot{allRobotIDs(1,1)}.poseCurrent(3) =
robot{allRobotIDs(1,1)}.poseCurrent(3) + 0.001;
                robot{allRobotIDs(1,1)}.poseCurrent(1) =
robot{allRobotIDs(1,1)}.poseCurrent(1) + 0.01;
%                 robot{allRobotIDs(2,1)}.poseCurrent(3) =
robot{allRobotIDs(2,1)}.poseCurrent(3) - 0.001;
                robot{allRobotIDs(2,1)}.poseCurrent(1) =
robot{allRobotIDs(2,1)}.poseCurrent(1) + 0.01;
              else

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
```

```
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                   % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                   % Update target pose.
                   MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
              end;

        elseif(F1 < F2 && (do1 > 2.3 && do2 < 2.3))

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_left(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.M
ovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
              for iMovObject = 1:nAllMovObjects
                % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                % Update target pose.
                MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
              end;
               if(err < 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                 for iMovObject = 1:nAllMovObjects
                % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                % Update target pose.
                   MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
```

```
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                    end;
%                      robot{allRobotIDs(1,1)}.poseCurrent(3) =
robot{allRobotIDs(1,1)}.poseCurrent(3)+ 0.001;
                    robot{allRobotIDs(1,1)}.poseCurrent(1) =
robot{allRobotIDs(1,1)}.poseCurrent(1) + 0.01;
%                      robot{allRobotIDs(2,1)}.poseCurrent(3) =
robot{allRobotIDs(2,1)}.poseCurrent(3) + 0.001;
                    robot{allRobotIDs(2,1)}.poseCurrent(1) =
robot{allRobotIDs(2,1)}.poseCurrent(1) + 0.01;
                else

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Move_right(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvData.
MovObjectVertex);%,globalEnvData.MovObjectPoseTarget);
                    for iMovObject = 1:nAllMovObjects
                       % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                       % Update target pose.
                       MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                    end;
                end;
            end;
        else
            [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
        end;
    else
           if(F1 > 0)

[globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent] =
forceformation(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvD
ata.MovObjectVertex,globalEnvData.MovObjectPoseTarget);
                for iMovObject = 1:nAllMovObjects
                  % Update current pose.

gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posCurrentHnd,Mo
vObject{allMovObjectIDs(iMovObject)}.dirCurrentHnd,MovObject{allMovObjectIDs
(iMovObject)}.poseCurrent,MovObject{allMovObjectIDs(iMovObject)}.shape,MovOb
ject{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iM
ovObject)}.radiusActual(2));
                    % Update target pose.
                    MovObject{allMovObjectIDs(iMovObject)}.poseTarget =
gui_move_pose('GetCoords',MovObject{allMovObjectIDs(iMovObject)}.posTargetHn
d);
%
gui_move_pose('Move',MovObject{allMovObjectIDs(iMovObject)}.posTargetHnd,Mov
Object{allMovObjectIDs(iMovObject)}.dirTargetHnd,MovObject{allMovObjectIDs(i
MovObject)}.poseTarget,MovObject{allMovObjectIDs(iMovObject)}.shape,MovObjec
t{allMovObjectIDs(iMovObject)}.polygonCoords,MovObject{allMovObjectIDs(iMovO
bject)}.radiusActual(2));
                end;
           else
```

```
        [globalEnvData.MovObjectVertex,globalEnvData.MovObjectPoseCurrent]
=
Stop_Movement(MovObject{allMovObjectIDs(iMovObject)}.poseCurrent,globalEnvDa
ta.MovObjectVertex);
       end;
end
```

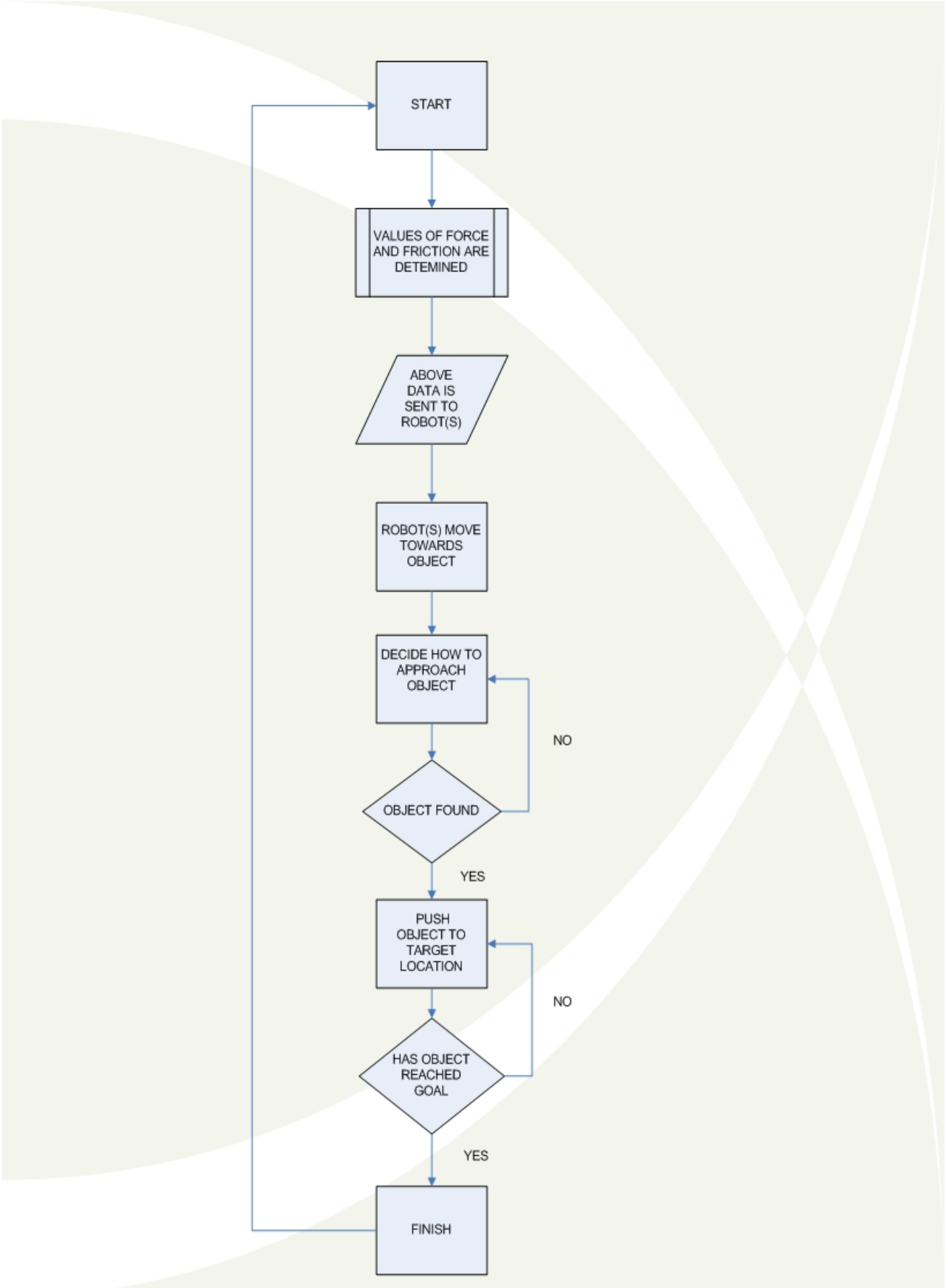
6.1.2 Mechanical torque function

```
  function [currentXNew,angVel1,anAcc1,TorqueRob1,angVel2,anAcc2,TorqueRob2]
= calculate_torq(poseCurrent,velCurrent,Mass,oldPos1,oldPos2)
%This program calculate the torque provided by each robot at any current
velocity at any time
% Mass = 1;
currentXNew(1,1) = poseCurrent(1) - oldPos1(1);
 currentXNew(1,1) = currentXNew(1,1)*pi;

 currentXNew(2,1) = poseCurrent(1) - oldPos2(1);
 currentXNew(2,1) = currentXNew(2,1)*pi;
% %  currentXNew
% %  currentXNew(2,1)
% %-------------------------------------------------------------------------
-
% %The main control of the robots,first the mass of the object is examined
by
% %the robots if the mass is equal to 0.5kg, one robot will be sent to push
an object, however
% %if the mass is greater than 0.5kg, two robots will do the job and this is
% %done in an if-else statement
%
% % -----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------
% %Calculating the torque applied by each robot
% %First angular velocity is calculated using the current velocity
% %(velCurrent) and then the angular acceleration is already given
% %robot wheel radius is 0.02 m, angular acceleration is 2.6704 rad.s-2
% %Mass of the object = 7kg A = 0.3, kinetic friction, angVel = v/r
%

angVel1 = (velCurrent/0.5);%ang velocity of robot 1
Inertia = (Mass*(pi/2));
anAcc1 = ((((0.4488).^2))-((angVel1).^2)  /(2*currentXNew(1,1)));
TorqueRob1 = ((Inertia*anAcc1) + (angVel1*0.3));

angVel2 = (velCurrent/0.5);%ang velocity of robot 2
anAcc2 = ((((0.4488).^2)) - ((angVel2).^2) /(2*currentXNew(2,1)));
TorqueRob2 = ((Inertia*anAcc2) + (angVel2*0.3));
% angVel1
% TorqueRob1% this robot one torque
% TorqueRob2% this robot 2 torque
```



Flow chart of how Program Basically functions