REMBOT: A RECONFIGURABLE, MINIATURE, MODULAR ROBOT WITH SOFT CONNECTION MECHANISMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

By
Mustafa UGUR
July 2023

ReMBot: A Reconfigurable, Miniature, Modular Robot with Soft Connection Mechanisms
By Mustafa UGUR
July 2023

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Onur Özcan(Advisor)

Yıldıray Yıldız

Melih Türkseven

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

REMBOT: A RECONFIGURABLE, MINIATURE, MODULAR ROBOT WITH SOFT CONNECTION MECHANISMS

Mustafa UGUR
M.S. in Mechanical Engineering
Advisor: Onur Özcan
July 2023

Nature has been a valuable source of inspiration for engineers, leading to the development of diverse materials, mechanisms, and algorithms that have enhanced human life. One fascinating idea borrowed from nature is the collaborative work of ant colonies. Ants work together to accomplish tasks that are impossible to achieve individually, such as constructing bridges by connecting to one another. Researchers have been motivated by such examples to create reconfigurable robots that can perform various exciting tasks, such as climbing stairs, crossing gaps, moving objects, and assisting in furniture building, by moving as separate modules and docking to each other using different connection mechanisms. However, the connection mechanism remains a challenge. Many of the existing designs require an actuator or a driving circuit which makes the control harder and limits the robot's motion.

This thesis presents ReMBot: A self-reconfigurable, miniature, modular robot with a soft connection mechanism. The robot comprises multiple modules, each equipped with backbones featuring permanent magnets. Using permanent magnets offers reconfigurability without requiring additional power, actuation, or a driving circuit while enhancing the robot's compliance. The modules, including the body, electronics, actuators, c-shaped soft legs, and backbones with magnets, weigh 29.43 $grams$ and have 82 mm x 60 mm x 14.7 mm dimensions. These module specifications, combined with the whole system design, allow ReMBot modules to execute path-tracking tasks, dock and undock, and sense the connection between modules. Their ability to connect and maintain a longer structure enables the ReMBot to climb obstacles higher than itself. Soft c-shaped legs enable modules to dock successfully by ensuring successful path-tracking tasks while they help them to move in different terrains like gravel, sand, or grass. The

modules' miniature structure, ease of manufacture, and affordability make them a suitable option for multiple use cases. The robot's wireless communication capability makes it a strong contender for surveillance in confined spaces like collapsed buildings and nuclear sites, large areas like farmlands, and even planetary exploration missions.

Keywords: Reconfigurable robots, miniature robots, soft robots, bio-inspired robots, legged robots, origami-inspired robots.

ÖZET

REMBOT: YUMUŞAK BAĞLANTI MEKANİZMALARINA SAHİP, YENİDEN-YAPILANDIRILABİLİR, MİNYATÜR, MODÜLER ROBOT

Mustafa UGUR
Makine Mühendisliği, Yüksek Lisans
Tez Danışmanı: Onur Özcan
Temmuz 2023

Doğa, insan hayatını geliştiren çeşitli malzemeler, mekanizmalar ve algoritmaların geliştirilmesine yol açarak, mühendisler için değerli bir ilham kaynağı olmuştur. Doğadan esinlenilen ilginç bir fikir, karıncaların ortak çalışmasıdır. Karıncalar, birbirine bağlanarak köprüler inşa etmek gibi, bireysel olarak başarmanın imkansız olduğu görevleri birlikte tamamlarlar. Böyle örneklerden ilham alan araştırmacılar olanüstü görevler gerçekleştiren bir çok yeniden-yapılandırılabilir robot geliştirmişlerdir. Bu ilginç görevler arasında merdiven çıkmak, boşlukları geçmek, nesneleri taşımak, ve hatta, mobilya oluşturmak bulunmkatadır. Bu görevleri birbirlerinden ayrı modüller halinde hareket edip istendiği zaman birleşme yetenekleri ile yerine getirmişlerdir. Tüm bu başarılı görevlere rağmen bu robotların birbirine bağlanmasını sağlayan mekanizmanın tasarımı hala bir sorun olarak varlığını sürdürmektedir. Var olan tasarımlar, kontrolü zorlaştıran bir eyleyici veya sürücü devre gerektirmekte ya da robotun hareketini sınırlamaktadırlar.

Bu tez, bir yeniden-yapılandırılabilir, minyatür, modüler bir robot olan ReM-Bot'u sunmaktadır. Robot, her biri mıknatıslara sahip omurgaları olan çoklu modüllerden oluşur. Omurgalarda bulunan mıknatıslar robotun esnekliğini arttırırken güç kullanımı, sürücü devresi ve ya herhangi bir eyleyiciye ihtiyaç duyulmadan robotun yeniden-yapılandırılabilir olmasını sağlar. Modüller, gövde, elektronik kart, eyleyiciler, c-şeklindeki yumuşak bacaklar ve mıknatıslara sahip omurgalar dahil olmak üzere $29,43$ $gram$ ağırlığında ve 82 mm x 60 mm x $14,7$ mm boyutlarındadır. Bu özellikler, sistem tasarımıyla birleştirildiğinde,

ReMBot modüllerinin yörünge izleme görevlerini gerçekleştirmesine, bağlanma ve ayrılmalarına, modüllerin aralarındaki bağlantıyı algılamasına olanak tanır. Modüllerin birbirleri ile bağlanarak dah uzun bir yapı oluşturma yeteneği ReMBot'un kendinden daha yüksek engelleri tırmanmasını sağlar. Yumuşak, c-şeklindeki bacaklar, modüllerin başarılı bir şekilde bağlanmasını sağlayarak çakıl, kum veya çim gibi farklı zeminlerde hareket etmelerine yardımcı olur. Modüllerin minyatür yapısı, kolay üretimi ve uygun maliyetli olması, modülleri çeşitli kullanım durumları için uygun bir seçenek haline getirir. Kablosuz iletişim kabiliyeti, çökmüş binalar ve nükleer siteler gibi sınırlı alanlarda, tarım arazileri gibi büyük alanlarda ve hatta gezegen keşif görevlerinde izleme için ReMBot'u güçlü bir aday yapar.

Anahtar sözcükler: Yeniden-yapılandırabilir robotlar, minyatür robotlar, yumuşak robotlar, doğadan esinlenen robotlar, origamiden esinlenen robotlar.

Acknowledgement

I want to give my deep appreciation who enabled this work. Firstly, I want to thank my advisor Professor Onur Özcan for allowing me to work at the Bilkent Miniature Laboratory. Our research together boosted my academic career and put me in a position I could not imagine. His expertise in robotics and deep knowledge were essential to my research for the last three years. Besides being a good advisor, he has always been a good friend and a great boss for me with his great sense of humor and understanding of work-related problems.

I am also thankful for my lab mates. They were always more than a coworker to me. I could not have had such a great time in the laboratory without all the humor. I will feel lucky to be worked with Nima Mahkam, Mert Ali İhsan Kalın, Amirali Abazari, Mohammad Askari, Alihan Bakır, Doğa Özbek, Muhammed Uygun, Yiğit Yaman, and Amirali Aleali.

The robot presented in this thesis cannot be possible without the help of undergraduate researchers on this project, especially Burak Arslan and Alperen Özzeybek. We worked really hard together and solved many problems mentioned in this thesis during the last three years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over generations, animals adapt their behaviors and physical appearances to suit the changing environmental conditions. Insect societies have been the subject of intriguing research into collective behaviors, revealing that despite lacking complex neural systems, some insects are capable of forming collective structures. These structures enable them to navigate obstacles, ensure the survival of colonies, and even create a microclimate for the benefit of their queen. [1].

Inspired by the intriguing collective behaviors of insects, robotics researchers have developed reconfigurable, miniature, and modular (RMM) robots [2]. These robots conduct tasks that are beyond their individual capabilities, such as climbing obstacles [3–6] and crossing gaps [7, 8], due to their reconfigurable structure. Moreover, their modular design allows for the creation of new structures or the adaptation of their locomotion capabilities to different tasks [9, 10]. With their ease of manufacturing, low cost, and exceptional operational capabilities, RMM robots have great potential for addressing challenging tasks for humanity, such as scanning in collapsed buildings, monitoring large farmlands, and even working on planetary exploration missions.

RMM robots rely heavily on their connection mechanism, which is responsible for linking their modules and creating a reconfigurable structure. Despite its crucial role, the connection mechanism is also the most challenging component to design. Miniature robots face size and power limitations that make it difficult to create a connection mechanism that operates without power consumption or requires additional driving circuits or actuators. Moreover, the connection mechanism should not impede the robot's movement and must align with its intended purpose. As a result, developing an effective connection mechanism for RMM robots is a complex and ongoing challenge for robotics researchers.

1.2 Literature Review and Research Aims

Over the past two decades, researchers have developed various types of RMM robots with innovative connection mechanisms for various tasks. The M-TRAN [10], developed for different tasks, uses a mechanical hook, which requires an additional DC Motor. Roombots [9], developed to build adaptive furniture, use a mechanical hook that needs an extra actuator, as well. Eciton robotica [4], designed to mimic ants, uses corkscrews for connection which requires an actuator. Puzzlebot [7] utilizes physical coupling without an actuator. However, this coupling creates a rigid connection limiting the robot's motion in different terrains. Other than robots with mechanical connections, many are equipped with magnets in their connection mechanism. SMORES-EP [3] uses electro-permanent magnets for connection which needs an auxiliary driving circuit for control. Also, its flat surface makes it impossible for one module to move other than a flat surface. Lily [11], developed to form modular structures in water, uses electro-permanent magnets as well. FreeBot [5], a sphere-shaped reconfigurable robot, utilizes permanent magnets and an iron shell for connection which makes it too heavy and impossible to move one module on slippery surfaces. Ant3DBot [8] and a myriapod robot [12] use an extra actuator to control the orientations of the permanent magnets. Modboat [13] utilizes a permanent magnet for connection and does not need any extra actuator. Yet, this robot's modules only move on the water surface, and there is limited collective use of it. FireAnt3D [6], and

FireAnt [14] applies current to heat the material in the docking part so that they melt and the modules stick to each other. Even though this method creates strong connections, it consumes too much power, and after some cycles, the material on the docking parts has to be changed.

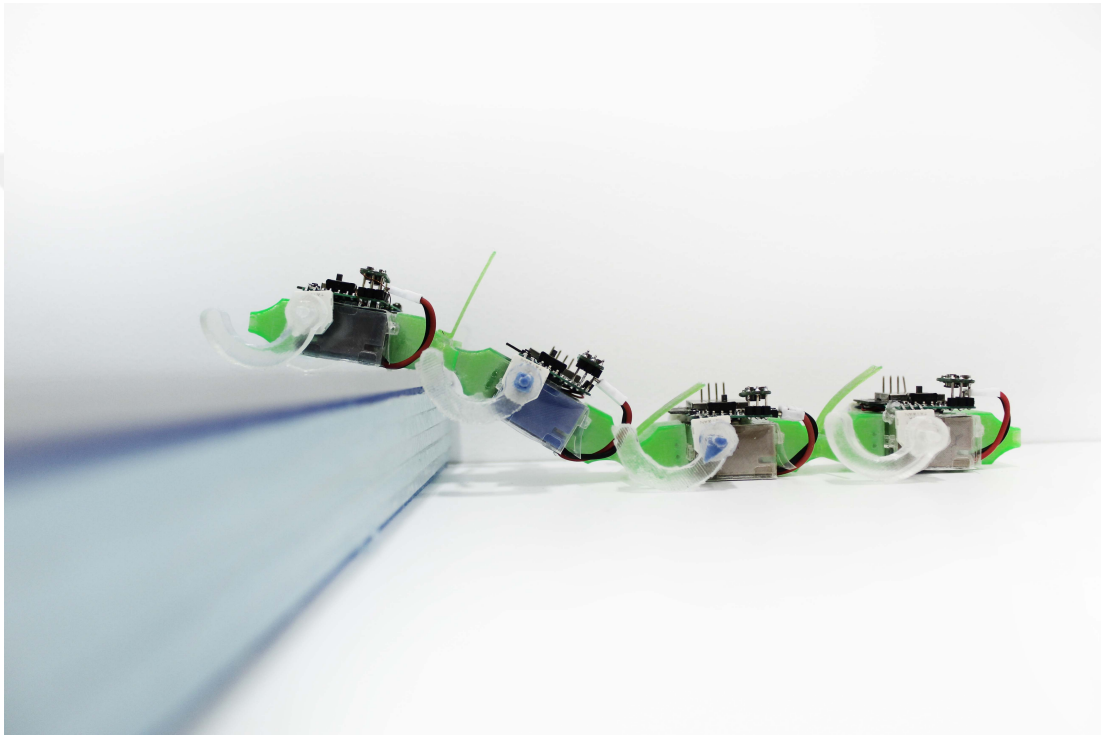


Figure 1.1: ReMBot: A reconfigurable, miniature, modular robot with a soft connection mechanism.

Although these robots perform remarkable tasks individually, they highlight the need for a more advanced connection mechanism. The connection mechanisms proposed in prior works necessitate additional actuators and driving circuits, leading to increased complexity in control. In addition, certain designs impede the robot's movement in diverse terrains, while others result in excessive weight. In this thesis, we propose ReMBot: A reconfigurable, miniature, modular robot with a soft connection mechanism, as shown in Fig. 1.1. The robot consists of palm-size modules, each weighing 29.43 *grams*. Our modules, equipped with soft c-shaped legs, can successfully conduct path-tracking tasks. The modules utilize cylinder-shaped permanent magnets as a connection mechanism, which does not require a particular actuator and does not consume power during the

operation. In addition to these advantages, this novel connection mechanism gives compatibility to the robot, which enables it to climb 37 mm obstacles despite its 14.7 mm height and move on different terrains like gravel, sand, or grass. The modules with soft c-shaped legs can perform path-tracking duties enabling them to dock each other. The mechanical design of the backbones gives an 80 % success rate in docking while giving a 100 % success rate in undocking. In general, the mechanical intelligence of the backbones removes the need for extra actuation and power consumption while helping the robot's motion by providing compliance.

1.3 Structure of the Thesis

The work done in this thesis is organized as follows.

Chapter 2 presents the design of the ReMBot. The design of the robot is split into two parts: system design and module design. The system design part mentions all the side systems enabling the reconfigurability of the robots. These are the motion capture system, main computer, server, and modules. The module design part describes the module's design in detail by talking about a module's components, which are the body, soft c-shaped legs, electronics, and backbones.

Chapter 3 gives a detailed description of the kinematics and dynamics models and the simulation environments to test those models. The kinematic modeling part mentions the model without considering the forces and torques acting on the robot body. It also presents the kinematics of the c-shaped legs and their difference from the standard wheel kinematics. It presents the motion equations according to the rotational motion of the c-shaped legs. The dynamic modeling part gives motion equations by considering the forces and torques acting on the body. Further, this part mentions the simulation environments we developed in MATLAB and Webots. This part also describes the tests we conducted on Webots to investigate the motion of one module and the climbing capabilities of different robot configurations.

Chapter 4 mentions the control of our robot. The control of our robot is divided into three parts in this chapter. First, the Linear Quadratic Gaussian (LQG) controller is described in detail. The LQG controlling the rotational velocities and rotational positions of the c-shaped legs work as an inner controller. In the second part, the path-tracking controller is given. This controller enables our modules to move to goal points, allowing them to dock with each other. Lastly, the synchronization algorithm is given, enabling our modules to move together when connected.

Chapter 5 gives a comprehensive explanation of the docking and undocking operations of the robot.

Chapter 6 gives the results of our experiments that we conducted to test the different abilities of our robot. The first part explains the effects of leg compliance on the path-tracking performance of the robot and the effects of leg choice for the obstacle climbing of one module. Then, the second part presents the effects of backbones on the performance of docking and undocking operations. The last part presents the effects of module numbers constituting the robot and the backbone type on the obstacle climbing performance.

Chapter 7 presents the case studies we conducted to demonstrate the reconfigurable structure and benefits of our robot. The first case study shows one module of ReMBot completing a sequential path-tracking task while climbing on an obstacle on its way. The second case study shows the reconfigurable use of our robot. Two modules conduct an obstacle climbing mission, while one module cannot do it previously. The last case study presents a similar case. In this case study, three modules complete an obstacle climbing task that two modules cannot.

Chapter 8 mentions the conclusions and the future works we plan to implement and try.

Chapter 2

Design

ReMBot is a reconfigurable, miniature, modular robot with a soft connection mechanism. The robot consists of modules that can dock/undock for conducting different tasks. Our robot owns this reconfigurability to its system design as a whole and its modules' design as individuals. The entire system makes it possible to locate modules, execute required path-tracking and provide wireless communication. On the other hand, each module helps this reconfigurability through its design.

2.1 System Design

The system comprises four main parts: modules, a motion capture system, a main computer, and a server. These four parts, with their own assignments, make our robot a reconfigurable robot that can conduct different duties. All of these four components work together, as shown in Fig. 2.1 to have a reconfigurable conditioning robot. Modules are the main components of our system with their miniature size. Along with this compactness, their origami-inspired design with basic electronics makes their mass production easy. Our system needs to know the poses of the modules running to give them as inputs to the path-tracking

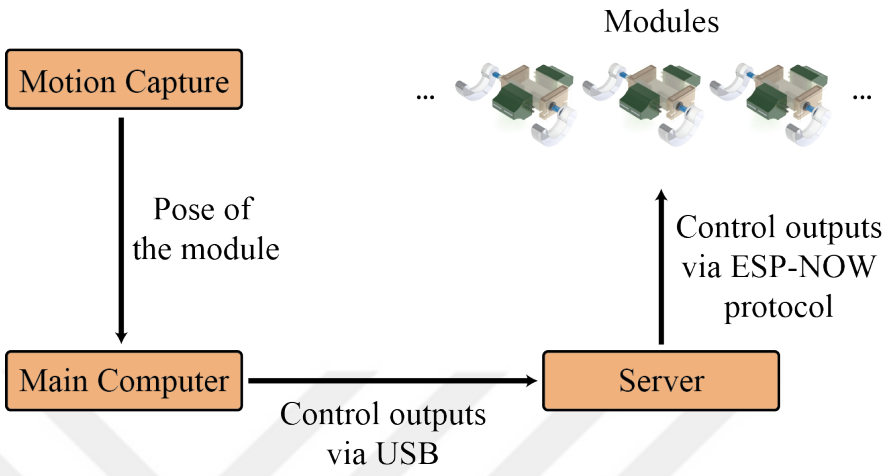


Figure 2.1: System design schematic to move the modules under the motion capture system.

algorithm. The motion capture system solves this problem and provides the poses of the modules under millimetric precision. The main computer gets the pose of the modules and executes the path-tracking algorithm, which is a feedback controller. Finally, the server takes these outputs from the main computer and passes them to modules wirelessly.

2.1.1 Modules

Modules are the main part of the whole system. Their body, electronics, soft c-shaped legs, and backbones enable them to serve as reconfigurable robots. They have two c-shaped legs connected to the body, and these legs provide the motion required to move the modules. The electronic boards on top of the modules control motion and communication. Backbones attached to the front and rear of the modules having permanent magnets keep modules together when they are connected.

2.1.2 Motion Capture System

The motion capture system (OptiTrack^{TM}) gets the position and orientation of the modules. We placed special markers on top of the modules to define rigid bodies. Then, the motion capture system provides data at 120 Hz via cable to the main computer. In addition to saving the data, the motion capture system can broadcast the data live using Software Development Kit(SDK) in Python. We used this SDK to get the poses of the modules live while modules are operating.

2.1.3 Main Computer

The main computer executes the path-tracking algorithm and gets the user inputs. It gets the poses of the modules and gives them as input to the path-tracking algorithm. The path-tracking algorithm uses a feedback controller to move modules to goal poses and outputs the required commands for each module, then transmits them to the server via USB.

The software can be configured to the need of the operation. For instance, the user can modify the system such that one module can move to another, dock, move together synchronously, and undock from each other. All these actions are defined as functions in the software previously. The user can select the order of these operations according to the need without deep programming knowledge. Also, a graphical user interface (GUI), shown in Fig. 2.2, helps the user to operate the robot. The user can start the operation, command the modules to dock/undock and stop the operation using this GUI. Also, the software always stores the position, orientation, time, controller inputs, and outputs for debugging purposes. Along with all the autonomous capabilities, the modules can be controlled by hand using the direction keys on the keyboard.

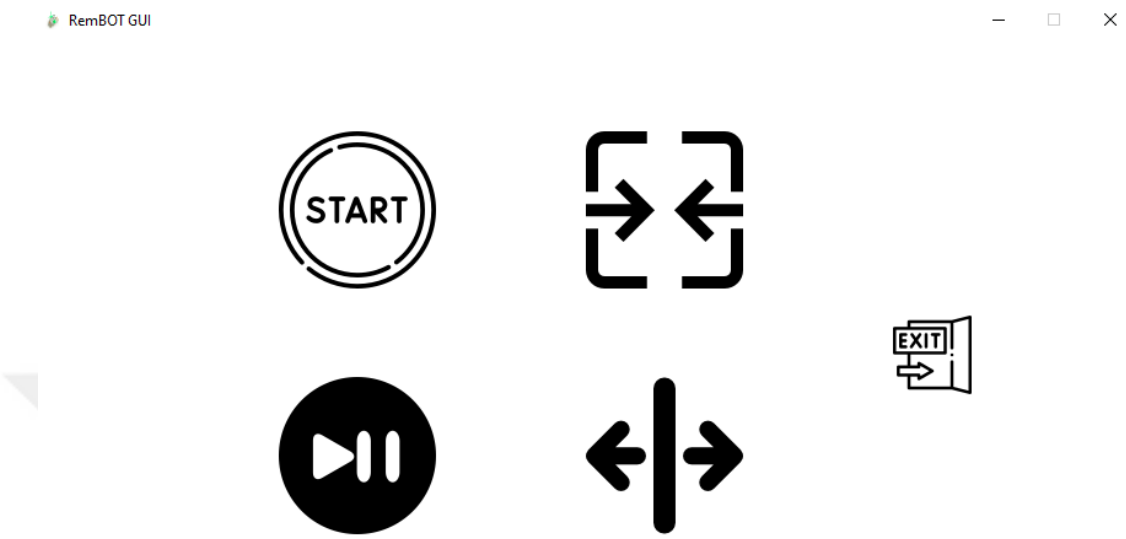


Figure 2.2: The Graphical User Interface (GUI) that allows users to control the operation.

2.1.4 Server

The server works as a messenger between the main computer and the modules. It gets the controller outputs from the main computer via USB and then transmits those data to the modules wirelessly. We used an external server instead of passing the controller outputs directly using the main computer. The essential reason for this is the communication issues between the main computer and the modules. For instance, transmitting data from the main computer to modules requires a wireless communication capability in the main computer. However, using a server removes this need.

We used a microcontroller unit (Esperessif Systems, ESP32-WROOM-32) for the server, the same microcontroller units (MCU) as our modules use for control. We selected these modules because they come with a built-in wireless communication capability. They support a lot of wireless communication protocols. One can use these boards to build internet-based applications easily. Also, they provide a

wireless communication protocol named ESP-NOW which offers a considerable advantage for developers. ESP-NOW is a very stable, well-documented protocol enabling ESP MCUs to transmit data easily and fast.

2.2 Module Design

Modules are the fundamental part of the whole system. These modules combine with a body, soft c-shaped legs, electronics, and backbones. One module, with all these, weighs 29.43 *grams*, while it has sizes of 82 *mm* in length, 60 *mm* in width, and 14.7 *mm* in height with legs considered. These modules with magnets in their backbones can dock with each other. At the same time, soft c-shaped legs enable undocking as well as climbing.

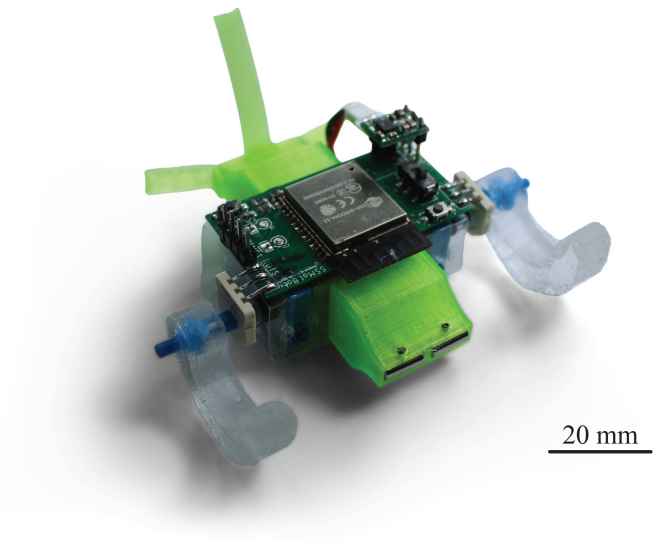


Figure 2.3: One module of ReMBot.

With the help of their design and the whole system structure, our modules together constitute a reconfigurable robot. These modules can move freely and scan a big area quickly while they can dock to each other and climb high obstacles that might be faced during these scanning duties. Their c-shaped soft legs enable them to climb obstacles while they also help them to move in different terrains such as

gravel, sand, or grass. On the other hand, their untethered design and wireless communication capability make them great candidates for operations that might be dangerous or tedious for people to work.

2.2.1 Body

The body is the essential part of the module, which holds all its components together. It combines an acetate sheet and some 3D-printed parts. The body's manufacturing starts with laser cutting the 2D design shown in Fig. 2.4. Then, manufacturing continues with folding the 2D design and putting 3D-printed parts in the required places in those folding patterns. These 3D-printed parts increase the structural integrity while keeping the actuators in place during the motion of the robot. After that, the DC motors, the battery, and the locks are placed, and the whole folded body is completed. The locks are manufactured using 3D printing of flexible Thermoplastic Polyurethane (TPU) filament. They connect the backbones to the body, and the user can connect them with backbones by placing pins.

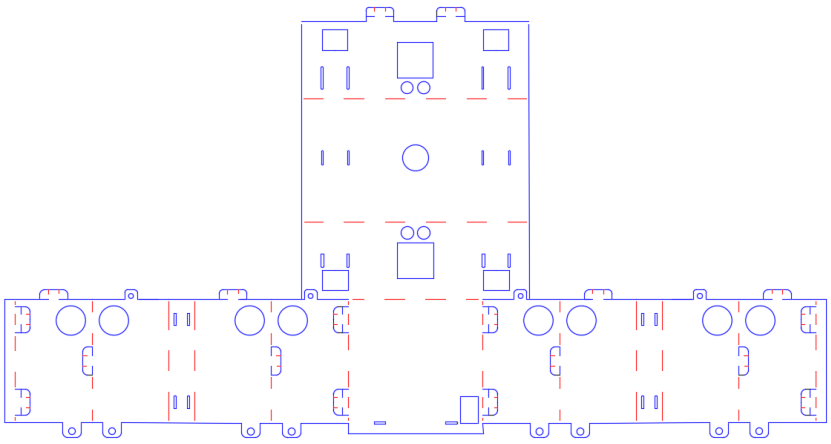


Figure 2.4: 2D design of the body.

2.2.2 Soft C-Shaped Legs

The modules use soft c-shaped legs to move. We selected soft c-shaped legs instead of standard wheels or rigid c-shaped legs because of their advantageous effects on the locomotion of the modules. They provide smooth trajectories while conducting path-tracking duties, which rigid c-shaped wheels cannot. This smoothness of soft legs has, also contributes to the docking success of the modules. Using standard wheels enables modules to have nearly the same trajectories with simulations and increases docking success. Nonetheless, using standard wheels makes it impossible for modules to move other than flat surfaces and climb obstacles. Legs have a diameter of 28 mm, which is bigger than the height of the folded body. This size difference enables higher obstacle climbing and successful undocking. Also, using soft material helps to undock by increasing the force between the leg and the ground.

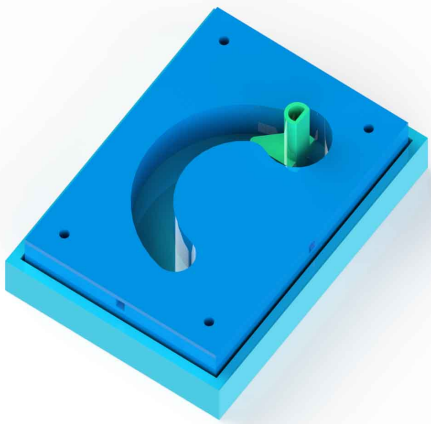


Figure 2.5: CAD illustration of the mold parts used in manufacturing soft c-shaped legs.

The c-shaped soft legs are manufactured using Polydimethylsiloxane (PDMS) by molding, and 3D-printed rigid shafts connect the DC motors with the legs. The manufacturing stars by 3D-printing the molds that are shown in Fig. 2.5. The mold is designed to consist of two parts to remove the molded part easily. After connecting these two parts with nuts, the rigid shaft shown in green is placed in its slot. Next, the user applies a silicone-based spray to remove the final product

easily. Then, the PDMS mixture is poured and placed in an oven for three hours in 60° Celsius. The user can separate the mold parts and get the soft leg after the curing.

2.2.3 Electronics

The electronics of the module are essential for two tasks which are controlling the motion of the module and the wireless communication. Our modules use a Linear Quadratic Gaussian (LQG) controller for the control of the gait and rotational velocity [15]. Even though our rotational sensors only measure the position, the LQG controller can control the rotational velocity with the help of the estimator inside it. Our modules communicate with each other and the server wirelessly. They get controller outputs from the server to maneuver, along with other commands, i.e., the start/stop signals and requests for docking/undocking. They also communicate with each other. For instance, each module sends a signal when another module is docked/undocked to it.

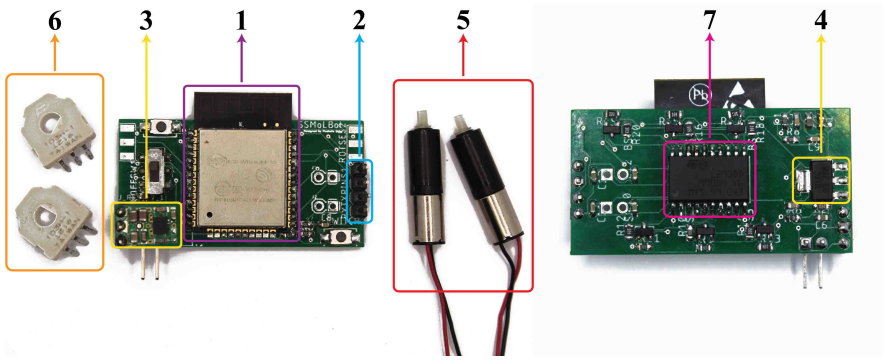


Figure 2.6: Electronics of one module: 1) ESP32 module, 2) programming pins, 3) 3.7V to 5V voltage regulator, 4) 5V to 3.3V voltage regulator, 5) DC motors, 6) potentiometers, 7) motor driver

We designed and manufactured a new printed circuit board (PCB) to handle both the control and the communication of the modules. The microcontroller unit (MCU) module (Esperessif Systems, ESP32-WROOM-32) on the PCB is the core component for these operations. This MCU module has built-in wireless

communication capabilities in addition to a well-established wireless communication protocol named ESP-NOW. Two rotational position sensors (Bourns, 3382) measure the position of the legs and gives them to the MCU, which sends the drive signals to the motor driver (STMicroelectronics, L293DD) according to the controller outputs. Then, motor drivers run the DC Motors (Polulu, 1:136 Sub-Micro Motor). All the electronics use a 150 mAh, 3.7V single-cell Li-Po battery as the power source. The step-up regulator (Polulu, S7V7F5) increases the voltage from 3.7V to 5.0V. On the other hand, a step-down regulator (Onsemi, NCP1117) decreases the 5V to 3.3V. While the DC Motors uses 5.0V, the MCU module uses 3.3V. We also added programming pins to change the embedded code inside the MCU in accordance with the changing operational needs.

We used DC motors because they are the best actuator meeting the sizes requirements. In our body, we only have only 1.5 cm x 2 cm space where we can place actuators. Also, their operation range between 0.5 Hz and 4 Hz enables use to control the motion of the robot. Lastly, their success in gait control makes them a great candidate for our robot [16, 17].

Figure 2.7: Backbone connecting modules, with soft extensions in sides, and two cylinder-shaped permanent magnets are placed.

2.2.4 Backbones

The backbone, shown in Fig. 2.7, is one of the crucial elements that make our robot reconfigurable with the permanent magnets in its end. They are manufactured from flexible TPU filaments using 3D printing. The holes at the end of them are designed to hold magnets tightly so that we do not need to glue them. Using permanent magnets for connection removes the need for an external driving circuit or an actuator while keeping the control problem simple. Also, these magnets only add 2 *grams* to the overall weight of one module. The chosen magnet design containing soft extension on its sides increases the success rates in docking and undocking operations. The cylinder-shaped magnets are strong enough to attack each other during the docking while allowing undocking. Additionally, their placement at the tip adds a rotational movement between modules which helps the robot during the motion.

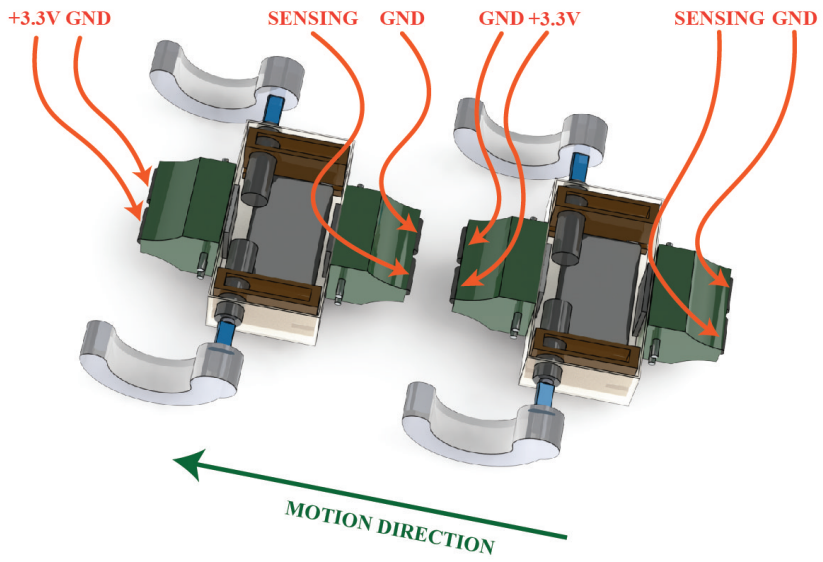


Figure 2.8: An illustration of two modules showing the electronic configuration of permanent magnets which enable them to be used as sensors for connection.

The magnets also work as a sensor which tells whether the module is connected to another module or not. The backbone on the rear side of the module works for sensing, while the backbone on the front side applies voltages, as shown in Fig. 2.8. In this configuration, one of the magnets is used for ground connections,

while the other magnet is used for sensing or applying voltage. When a module comes from the back and docks to another module, the module in the front knows that one module is docked to itself. Then the module in the front sends a wireless synchronization signal to the rear module to walk together.

Chapter 3

Modeling

Mobile robots, in a general perspective, can be modeled in different ways. Kinematic models are used to model mechanisms with relatively simple motions. For instance, the kinematic modeling of a bicycle, tricycle, or differential-drive robot can sufficiently give a perspective about the motion of these moving objects. However, when too many interactions involve the motion of the robot and when someone wants to have a better intuition about the motion of the object, dynamic modeling arises [18].

We started the modeling by creating some kinematic models of our robot. In the kinematic models, we first assumed our robot was a differential-drive robot with standard wheels. Then, we improved our model by adding c-shaped hemispherical legs. After seeing the difference between the experiments, we moved to dynamic models to have a better representation of the robot's motion in the simulation environment.

3.1 Kinematic Modeling

ReMBot uses two c-shaped(hemispherical) legs for locomotion. The simplest way to model its motion is by modeling the kinematics as a differential-drive robot

with wheels. Using the differential-drive model gives the user an intuition about the motion of the robot for given actuator inputs. Also, the feedback controllers developed based on this model make modules successful in their path-tracking duties.

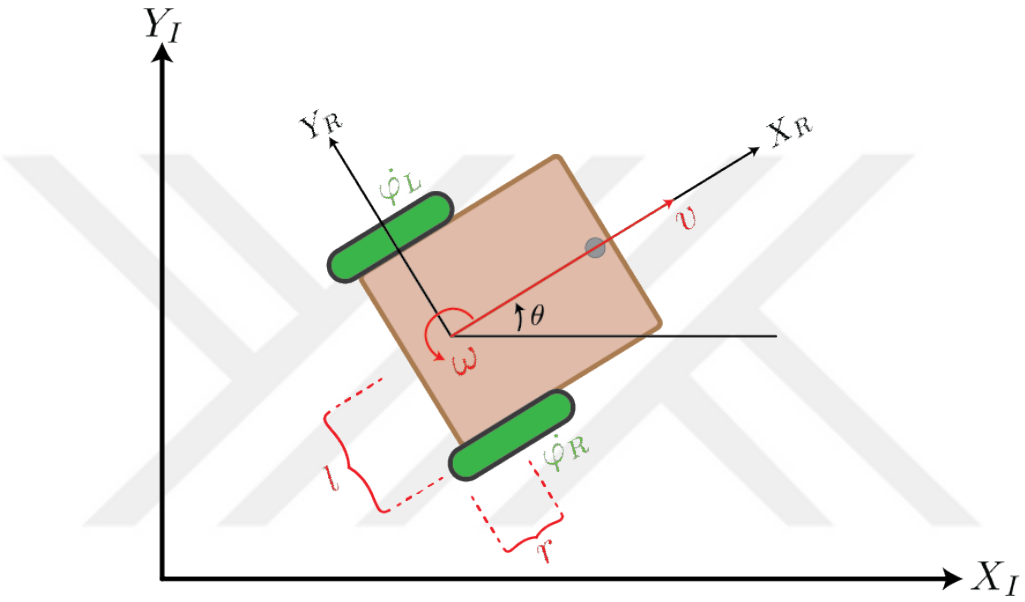


Figure 3.1: A differential-drive robot illustrated in the inertial frame.

Even though the differential-drive model is adequate for developing the feedback controller for the path-tracking duty, there are some essential differences between the motion of a c-shaped legged robot and a differential-drive robot. For a wheeled robot, the robot has a linear velocity (v) that can be defined as follows:

$$v = r\dot{\varphi} \tag{3.1}$$

where r is the radius of the wheel and $\dot{\varphi}$ is the rotational velocity of the both wheels. In this kind of motion, because there will be no turning, the rotational velocity of the robot will be zero.

In differential-drive robots, like the robot shown in Fig. 3.1, we need to calculate the rotational velocity (ω) as well. The forward kinematics of the differential-drive robots can be represented as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\varphi_R} \\ \dot{\varphi_L} \end{bmatrix} \tag{3.2}$$

where $R(\theta)$ rotation matrix can be defined as:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

where $\dot{x}$ is the translational velocity (v), θ is the orientaion of the robot according to the intertial frame I. $\dot{\theta}$ is the rotational velocity (ω), r is the radius of the wheels, $2l$ is the distance between the wheels, and $\dot{\varphi_R}$ and $\dot{\varphi_L}$ are the right and left wheels' rotational velocities respectively.

By further calculations, Eq. 3.2 can be written as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{r\dot{\varphi_R}}{2} + \frac{r\dot{\varphi_L}}{2} \\ 0 \\ \frac{r\dot{\varphi_R}}{2l} - \frac{r\dot{\varphi_L}}{2l} \end{bmatrix} \tag{3.4}$$

$$\begin{aligned} \dot{x_I} &= \cos(\theta) \left(\frac{r\dot{\varphi_R}}{2} + \frac{r\dot{\varphi_L}}{2} \right) \\ \dot{y_I} &= \sin(\theta) \left(\frac{r\dot{\varphi_R}}{2} + \frac{r\dot{\varphi_L}}{2} \right) \\ \dot{\theta_I} &= \frac{r\dot{\varphi_R}}{2l} - \frac{r\dot{\varphi_L}}{2l} \end{aligned} \tag{3.5}$$

For developing the inverse kinematics, one needs to know the translational velocity $(\dot{x}, v)$, and the rotational velocity $(\dot{\theta}, \omega)$, which are defined in the following equation:

$$
\begin{aligned}
\dot{x_R} &= \frac{r\dot{\varphi_R}}{2} + \frac{r\dot{\varphi_L}}{2} \\
\dot{y_R} &= 0 \\
\dot{\theta_R} &= \frac{r\dot{\varphi_R}}{2l} - \frac{r\dot{\varphi_L}}{2l}
\end{aligned}
\tag{3.6}
$$

After getting these parameters the rotational velocity of right wheel($\dot{\varphi_R}$) and the rotational velocity of the left wheel ($\dot{\varphi_L}$) can be calculated as follows:

$$
\begin{aligned}
\dot{\varphi_R} &= \frac{\dot{x_R} + l\dot{\theta_R}}{r} \\
\dot{\varphi_L} &= \frac{\dot{x_R} - l\dot{\theta_R}}{r}
\end{aligned}
\tag{3.7}
$$

As it can be seen from the calculations above, the radius of the wheel(r) is constant. However, a c-shaped leg does not make contact with the ground as exactly as a wheel. The radius changes over time while the c-shaped leg moves. There are three motion periods of a c-shaped leg, namely rolling, overturning, and pure rotation, as shown in Fig. 3.2. For each of these periods, the effective radius must be calculated [19].

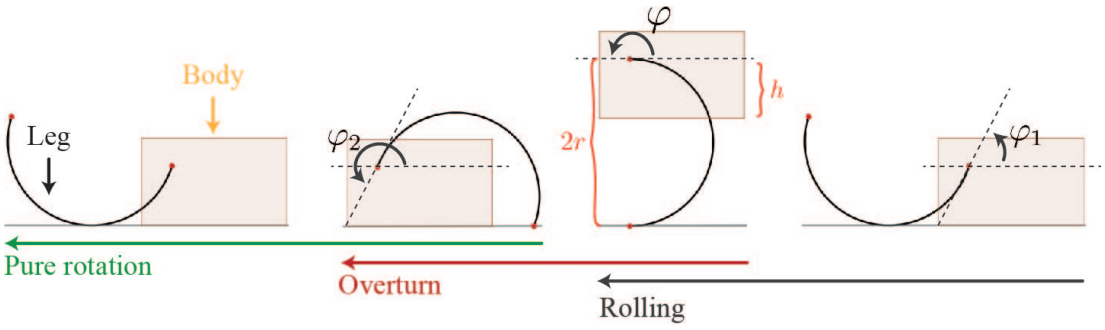


Figure 3.2: The motion of a module with the c-shaped leg. While the module moves toward right rolling, overturn and pure rolling phases of the leg's movement are shown.

In rolling motion, the leg slip on the ground like a wheel. However, the leg's center of rotation is not located in the center of the vertical center of the module. Also, the diameter ($2r$) of the c-shaped leg is bigger than the height (h) between

the center of rotation and the bottom part of the module. With consing these two reasons, we can represent the Eq. 3.1 as follows:

$$v = (r - r\cos\varphi)\dot{\varphi} = r(\varphi)\dot{\varphi} \tag{3.8}$$

where φ is the rotation position angle of the leg and $r(\varphi)$ is the effective radius.

For the rolling motion, the starting angle φ_1 can be calculated from the body height h and semicircle radius r as follows:

$$\varphi_1 = \arccos\left(\frac{r-h}{r}\right) \tag{3.9}$$

which in our current design $\varphi_1 = 77.16°$, with $2r = 28\ mm$ and $h = 10.89\ mm$.

The rolling motion stops when the contact point with the ground coincides with the end of the hemispherical leg $\varphi = 180°$. After φ reaches 180°, the c-shaped leg starts to rotate around the contact point until $\varphi = \varphi_2$. The equation of motion for the overturn period of the c-shaped leg can be represented as follows:

$$v = -2r\cos\varphi\dot{\varphi} = r(\varphi)\dot{\varphi} \tag{3.10}$$

Similar to the previous case the angle φ_2 can be calculated from the dimensional parameters of the robot as follows:

$$\varphi_2 = 270° - \arcsin\left(\frac{h}{2r}\right) \tag{3.11}$$

which in our current design $\varphi_2 = 247.11°$.

In the final portion of the movement, the c-shaped leg simply rotates without propelling the body forward, which makes the linear velocity and effective radius zero.

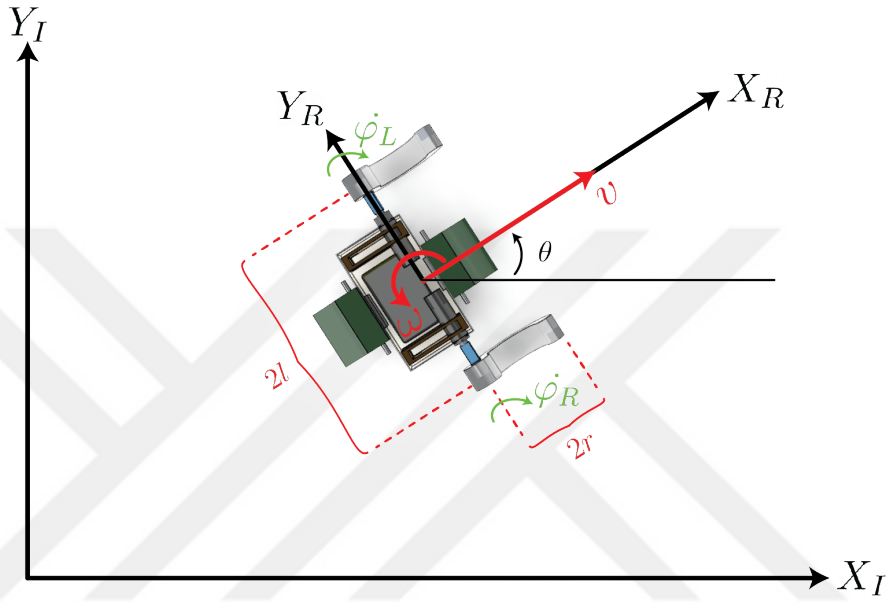


Figure 3.3: A ReMBot module in the inertial reference frame.

After calculating the effective radius for all motion periods, we can describe the forward kinematics of our modules as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r(\varphi_R)}{2} & \frac{r(\varphi_L)}{2} \\ 0 & 0 \\ \frac{r(\varphi_R)}{2l} & -\frac{r(\varphi_L)}{2l} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} \tag{3.12}$$

By further continuing the calculations in Eq. 3.12, we can write the forward equations for varying radius assumptions like the following:

$$
\begin{aligned}
\dot{x_I} &= \cos(\theta)\left(\frac{r(\varphi_R)\dot{\varphi_R}}{2} + \frac{r(\varphi_L)\dot{\varphi_L}}{2}\right) \\
\dot{y_I} &= \sin(\theta)\left(\frac{r(\varphi_R)\dot{\varphi_R}}{2} + \frac{r(\varphi_L)\dot{\varphi_L}}{2}\right) \\
\dot{\theta_I} &= \frac{r(\varphi_R)\dot{\varphi_R}}{2l} - \frac{r(\varphi_L)\dot{\varphi_L}}{2l}
\end{aligned}
\tag{3.13}
$$

For developing the inverse kinematics, like in the constant radius case, one needs to know the translational velocity $(\dot{x}, v)$, and the rotational velocity $(\dot{\theta}, \omega)$, which are defined in the following equation:

$$
\begin{aligned}
\dot{x_R} &= \frac{r(\varphi_R)\dot{\varphi_R}}{2} + \frac{r(\varphi_L)\dot{\varphi_L}}{2} \\
\dot{y_R} &= 0 \\
\dot{\theta_R} &= \frac{r(\varphi_R)\dot{\varphi_R}}{2l} - \frac{r(\varphi_L)\dot{\varphi_L}}{2l}
\end{aligned}
\tag{3.14}
$$

After getting these parameters the rotational velocity of right wheel$(\dot{\varphi_R})$ and the rotational velocity of the left wheel $(\dot{\varphi_L})$ can be calculated as follows:

$$
\begin{aligned}
\dot{\varphi_R} &= \frac{\dot{x_R} + l\dot{\theta_R}}{r(\varphi_R)} \\
\dot{\varphi_L} &= \frac{\dot{x_R} - l\dot{\theta_R}}{r(\varphi_L)}
\end{aligned}
\tag{3.15}
$$

An important point to note about the inverse kinematic equations is that in pure rotation, $r(\varphi) = 0$. Therefore, the calculations for the rotation rate of the leg result in infinity. To solve this issue, a maximum frequency for the leg rotation is defined during the simulations.

Using this forward kinematics calculation with varying radius values instead of fixed radius values helps us to have a better model for the kinematics of our modules. In this assumption, the translational velocity becomes a function of the position and the rotational velocity of the leg. However, in the standard wheel assumption, the translational velocity is only a function of the rotational velocity

of the wheel. Also, having the position of the leg in the function makes it possible to model the different gaits of the module, i.e., trot gait where the left leg and right legs have 180° phase difference.

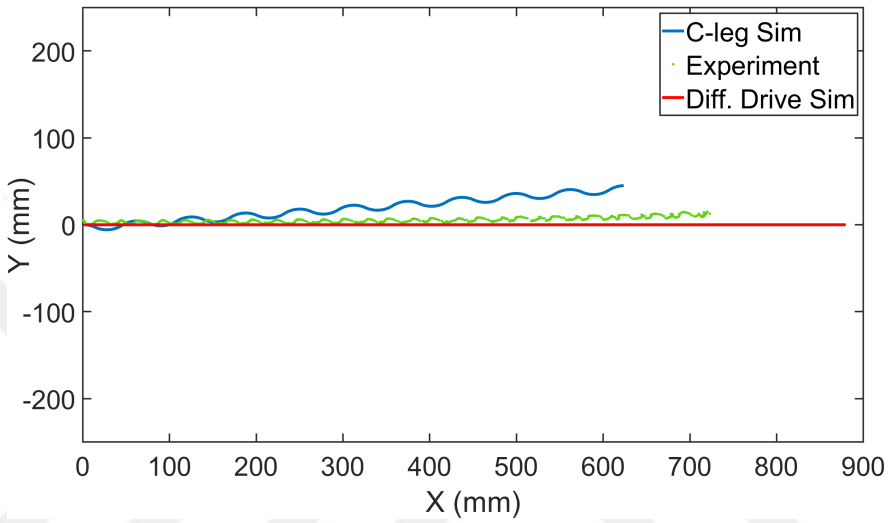


Figure 3.4: Forward kinematics simulations for the robot.

We developed two different kinematics simulations using MATLAB after we developed these mathematics. We first developed the simulation based on the constant radius assumption, accepting our robot is a simple differential-drive robot with standard wheels. Then, because this model was not sufficient to model the motion of our modules, we developed another simulation based on the varying radius assumptions given in Eq. 3.12. Fig. 3.4 shows the differences between these two assumptions and the real robot running on trot gait. Both the simulations and the experiment are conducted for a 5 *seconds* period on a flat table using a ReMBot module operating on 2 Hz. We simulate a robot moving in trot gait where two legs have 180° *degrees* phase difference. The blue line, showing the simulation result for varying radius assumptions, can represent the trot gait moving toward the upside. The red line, on the other hand, shows the simulation result for the differential-drive robot assumption. The green dots show the experiment results for a module running on trot gait.

Table 3.1: Speed results for simulations and experiment for a ReMBot running on trot gait.

Configuration	***Speed*** (mm/s)
C-Shaped - Simulation	125
Diff-Drive - Simulation	175.57
C-Shaped - Experiment	144.62

Table 3.1 shows the speed results for both the simulations and the experiment shown in Fig. 3.4. These results show that the simulation using c-shaped kinematics gives better results for inspecting the speed values of the modules. However, the motion patterns in the simulation using c-shaped kinematics and the experiment for the c-shaped leg are not matching.

Along with this mismatch, there is a need to simulate the robot consisting of multiple modules. The cylinder-shaped backbones create a rotation between the modules, which we cannot model in our simple kinematic simulation environment, which brings us to build a dynamic simulation environment.

3.2 Dynamic Modeling

The modular structure of the ReMBot obliges us to use a dynamic simulation environment. The backbones connecting the modules use permanent magnets. These permanent magnets, with their cylinder shape, provide the required attraction force to hold modules together. On the other hand, the cylindric shape causes a rotational motion shown in Fig. 3.5. The rotational motion enables the robot to comply with the ground it moves on. For instance, when the robot tries to climb an obstacle, the robot's overall shape looks like the obstacle. This compliance allows the robot to climb obstacles better while it also helps it to move on another surface, such as gravel, soil, or grass.

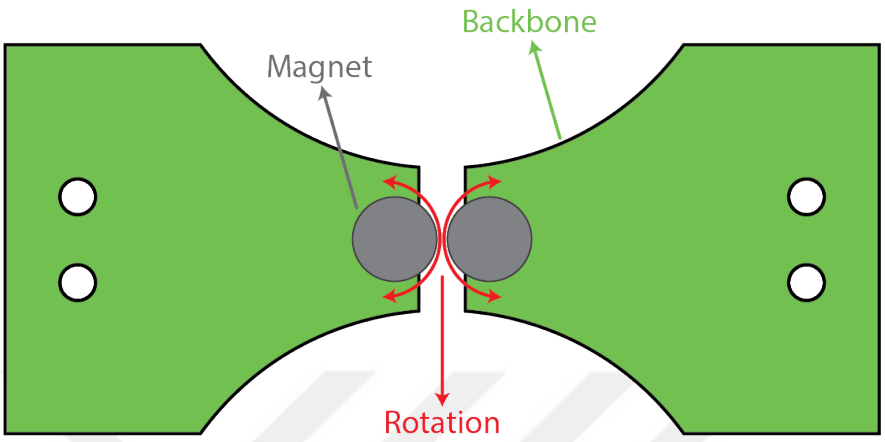


Figure 3.5: Backbones with cylinder-shaped magnets shown with the rotation around the magnets.

Even though the rotation between the modules is one of the most characteristic and beneficial movements happening on the robot, the kinematic simulation environment is inadequate to model this behavior. Other than this behavior, we observed that the robot's motion is dependent on the surface it moves. Because of these two main reasons, we decided to simulate our robot dynamically by involving the force and torque calculations. Also, when we used more than one module, the kinematic modeling of our robot diverged from the differential-drive assumption and acted like a skid-steer robot which requires complex kinematic calculations. In this skid-steer case, the forward kinematics calculations change, and one has to find some constant with experiments [20].

Dynamic simulations, different than kinematic simulations, involve the forces and torques acting on the body. The forces and moments acting on the body mainly depend on the ground the robot moves. For instance, the robot moves faster when it moves on a velvet textile surface than when it moves on a flat wood surface. This happens because the legs have a bigger grip when moving on velvet.

In dynamic simulation developments, we first used the existing simulation we developed in MATLAB for our miniature robots. Then we moved our simulations to Webots, which helped us try new configurations quickly.

3.2.1 Dynamic Simulations on MATLAB

We first employed our existing dynamic simulation developed in MATLAB. This simulation environment was developed to simulate our previous miniature robots, MinIAQ-III and SMoLBot. This simulation environment is first developed based on the rigid body dynamics of the MinIAQ-III, and it's effectiveness in modeling the motion of the miniature robots is shown in our previous works [15, 17, 21].

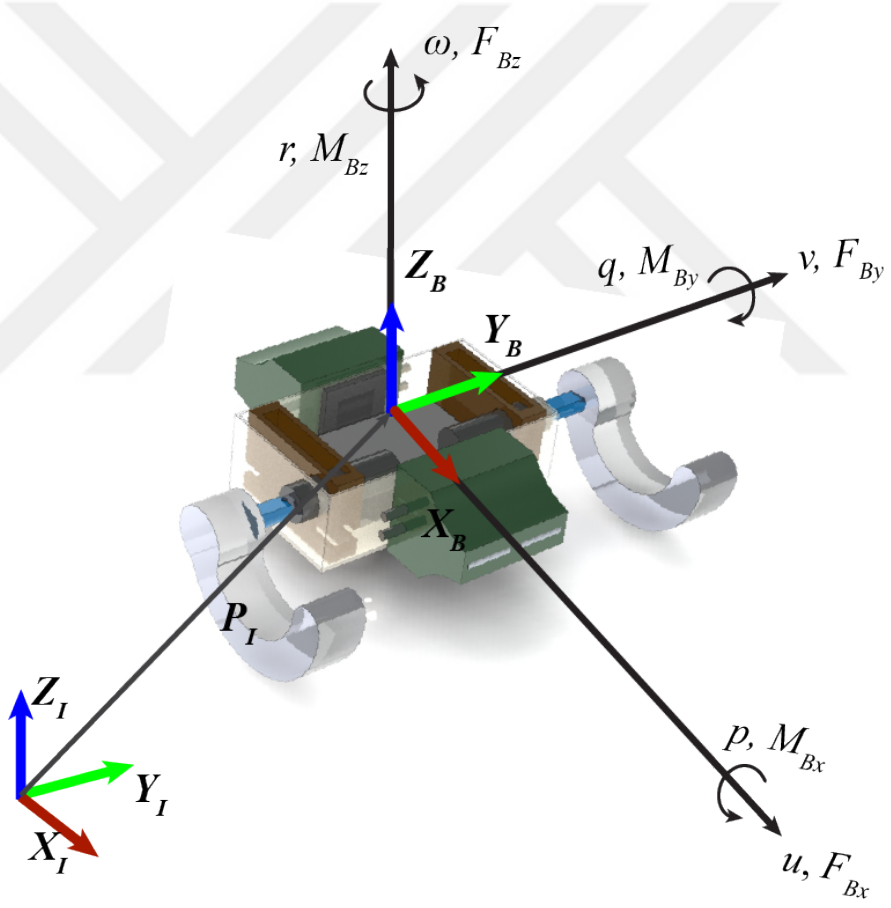


Figure 3.6: Schematic representation of one module ReMBot with body-attached and inertial reference frames used in deriving the dynamic equations

The simulation uses rigid body equations shown in Eq. 3.16. In this simulation, a state vector of six translational and six rotational variables is solved in very small time steps using dynamic equations and the stiff differential equation solvers in MATLAB. Fig. 3.6 shows one module of ReMBot with parameters used to develop dynamics. The body-attached axes (B) are placed at the center of mass

of the robot's frame and are used to define the orientation based on Euler angle rotations (roll ϕ, pitch θ, and yaw ψ). The body's position is also tracked over time with respect to an inertial reference frame (I) fixed to the ground level and initially parallel with the body-attached frame.

$$\begin{pmatrix} \dot{P}_{Ix} \\ \dot{P}_{Iy} \\ \dot{P}_{Iz} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} (c\theta c\psi)u + (s\theta s\phi c\psi - s\psi c\phi)v + (s\theta c\phi c\psi + s\psi s\phi)w \\ (c\theta s\psi)u + (s\theta s\phi s\psi + c\psi c\phi)v + (s\theta c\phi s\psi - c\psi s\phi)w \\ (-s\theta)u + (c\theta s\phi)v + (c\theta c\phi)w \\ p + (q\sin\phi + r\cos\phi)\tan\theta \\ q\cos\phi - r\sin\phi \\ \dfrac{q\sin\phi + r\cos\phi}{\cos\theta} \\ \dfrac{F_{Bx}}{m} - (qw - rv) \\ \dfrac{F_{By}}{m} - (ru - pw) \\ \dfrac{F_{Bz}}{m} - (pv - qu) \\ \dfrac{M_{Bx} - (I_{zz} - I_{yy})qr}{I_{xx}} \\ \dfrac{M_{By} - (I_{xx} - I_{zz})rp}{I_{yy}} \\ \dfrac{M_{Bz} - (I_{yy} - I_{xx})pq}{I_{zz}} \end{pmatrix} \tag{3.16}$$

where $\boldsymbol{P_{\mathcal{I}}} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$, $\boldsymbol{\Gamma} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top$, $\boldsymbol{V_B} = \begin{bmatrix} u & v & w \end{bmatrix}^\top$, and $\boldsymbol{\Omega_B} = \begin{bmatrix} p & q & r \end{bmatrix}^\top$ define the position, orientation, translational speed, and rotational velocity of the robot, respectively. Note that c and s are abbreviations for cosine and sine functions, respectively, m denotes the mass of the robot, and $\{I_{xx}, I_{yy}, I_{zz}\}$

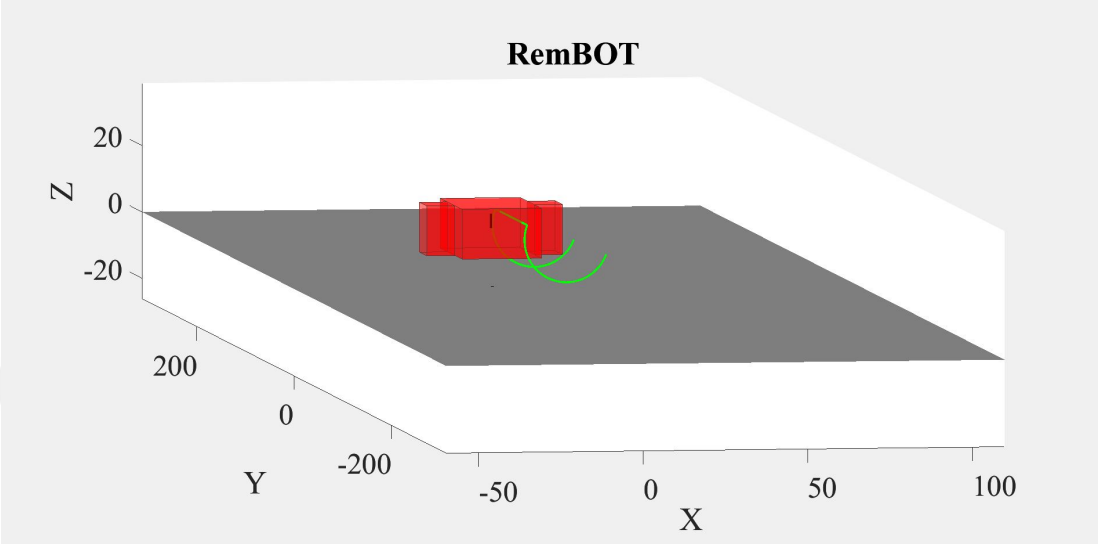


Figure 3.7: The dynamic simulation environment developed in MATLAB. The red body is the body, and the green hemispherical shapes are the legs.

represent the mass moments of inertia of the robot's rigid body about the body-attached axes. The inputs to the system are the net external force and moment ($\boldsymbol{F_B}$ and $\boldsymbol{M_B}$) along the body-attached axes, which are determined by a non-linear viscoelastic spring-damper model based on feet-ground interactions. The contact force model is an implicit function of all system states as well as the leg kinematics. The detailed derivation of the dynamics and the contact force model are presented in [21].

The net force and moment acting on the body frame must be known to solve the Eq. 3.16. The ground reaction force in our simulations is defined by Hunt and Crosley's model which calculates the ground reaction force as a function of penetration length (δ) and penetration rate ($\dot{\delta}$) as follows:

$$F_{ground} = K_{spring}\delta^n + C_{damper}\dot{\delta} \tag{3.17}$$

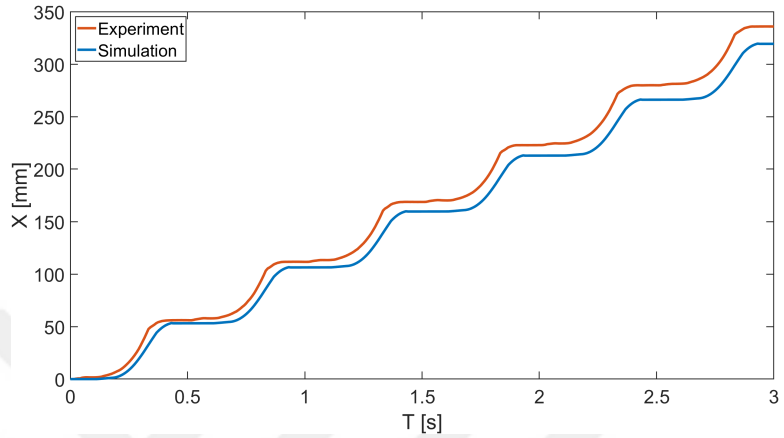
where F_{ground} is the ground reaction force acting on the leg, K_{spring} is the spring coefficient, n is the nonlinear exponent, and C_{damper} is the damper coefficient. In

our simulation, the coefficients K_{spring} and C_{damper} are found by running an optimization. First, we collected translational velocity data on experiments. Then, a Genetic Algorithm program in MATLAB was utilized to get the best-fitting physical parameters. This process is used instead of writing the coefficients directly because the physical parameters of the materials were not known certainly, and they change with the ground material where we run experiments.

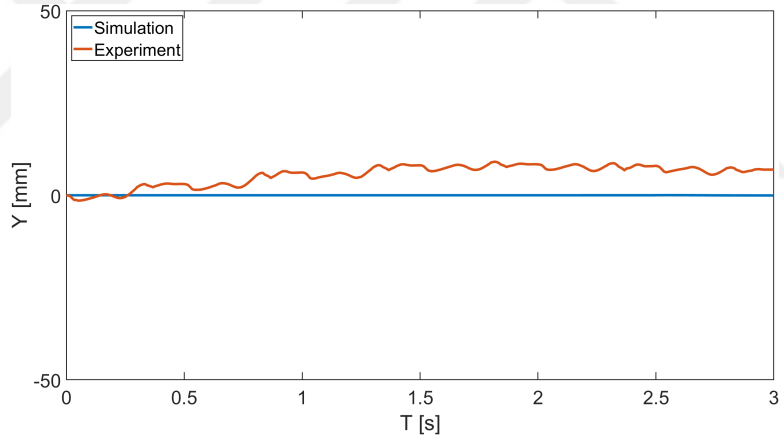
The existing environment was not containing the ReMBot modules. We first implemented our modules in the simulation by hand-coding the body and leg geometries and actuation parts, as shown in Fig. 3.7. The picture is taken as a screen shoot from our simulation. The red parts represent the body and backbones together, while the green hemispherical shapes are the c-shaped legs.

We investigated the correctness of the simulations by comparing the experimental results with the simulation result. Fig. 3.8(a) shows the position of one module on the x-axis, Fig. 3.8(b) shows the position of the module on the y-axis, and Fig. 3.8(c) shows the position of the module on the z-axis. The experiments were conducted with 2 Hz rotational velocity on a flat wood table for 3 *seconds*. The simulations were also executed for 3 *seconds*. All of these figures prove that the simulation environment is sufficient to model the motion of the modules.

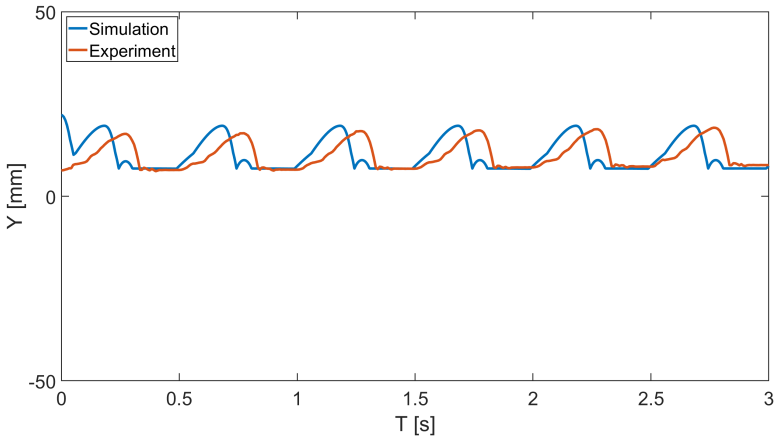
The main disadvantages of the simulation environment on MATLAB were its slow execution and its complex structure. Depending on the defined gait and the rotational velocity of the legs, the simulation run time can be quite big. For instance, our computer (Intel i7-7700 CPU, 64GB RAM) needs nearly 4 *minutes* to simulate a 3 *seconds* motion of one module. The long contact surface on the hemispherical leg and the big contact area of the module's bottom part makes it harder to calculate the impact forces defined in Eq. 3.17, which slows down the whole dynamic calculation. On the other hand, the complex structure of this simulation environment makes it harder to add another robot configuration, such as a ReMBot with four modules. One needs to hand-code all the geometry of the new robot. This hand-coding part adds a substantial time, even for an experienced programmer.



(a)



(b)



(c)

Figure 3.8: Motion of one module on MATLAB and experiment.

3.2.2 Dynamic Simulations on Webots

Our simulation environment's long run time and complex structure led us to use an existing robot simulation environment. Using existing robot simulators provides great advantages over hand-written simulators. They generally come with different physics engines that can run the dynamics very quickly. Also, one can import robots from the design program in different file formats and then can use those files to import robots to the simulation environment. Using these files to have a robot in simulation instead of hand-coding them enables developers to spend their time on more important things.

There are plenty of robot simulators such as Webots, CoppeliaSim, or Gazebo. Both three of them are free for educational use and have many tutorials on the web. We chose Webots because it is the easiest one to import robot designs. Webots is a free and open-source robot simulator. One can export robot designs from CAD software (i.e., SolidWorks, Fusion 360) as VRML files and import them to Webots. Webots uses Open Dynamics Engine (ODE) for making the dynamics calculations of the rigid bodies and enables users to model the forces and moments accurately. The user can add different actuators and sensors on the robot, such as GPS sensors, odometers, lidars, cameras, or light sensors, and simulate the input received from those sensors. Also, one can write a controller in C++, Python, or MATLAB and implement that controller using an Application Programming Interface (API).

For the simulation of the ReMBot in Webots, we first need to export the VRML file from the CAD software (Solidworks). For each configuration of the robot, the design files must be exported separately. For instance, the ReMBot consisting of two modules and the ReMBot consisting of four modules must be exported as separate VRML files. Then, we define rigid bodies and joints. In our case, one module has three rigid bodies. The first rigid body consists of the foldable body, motors, battery, backbones, and magnets. The second and third rigid bodies are the right and left legs, respectively. After defining the rigid bodies, we define the joints and joint types. Then, we put actuators assigned to those joints. For our

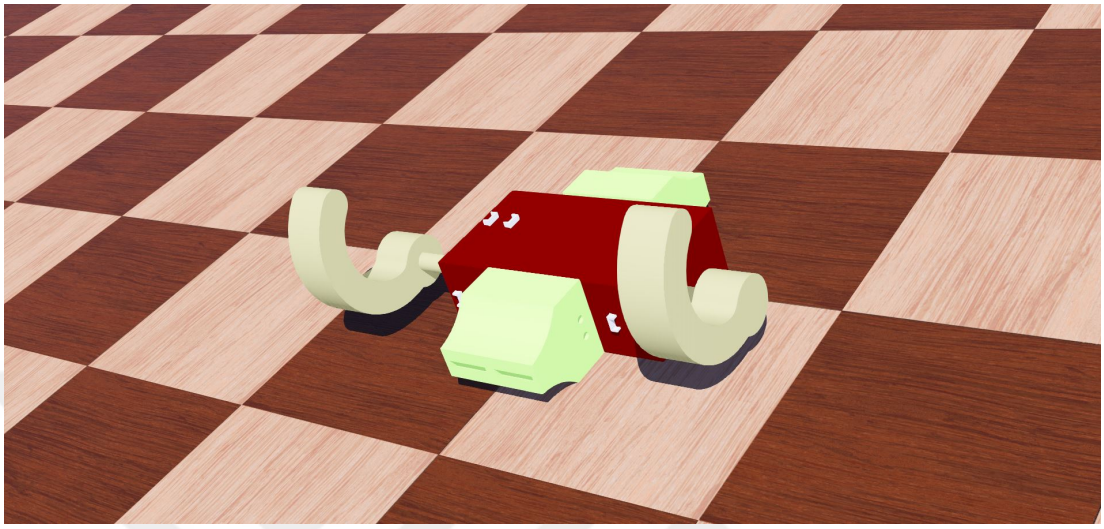


Figure 3.9: One module of ReMBot in Webots robot simulator.

modules, we picked rotational joint types and used rotational motors as actuators. With all these setups, one module of ReMBot in the Webots robot simulator is shown in Fig. 3.9.

After implementing the robot and making the required arrangements, we wrote a simple controller in Python to control the rotational velocities of the legs. Using this controller and the robot model, we compared the simulation and the experiment results. Different than our simulation in MATLAB, we used a module operating in 1 Hz. We changed the rotational velocity we investigated because we saw that using lower frequencies in operation is more beneficial, as discussed in Sec. 6. We compared the experiments and simulation for a 3 *seconds* duration and conducted them on a flat wood surface.

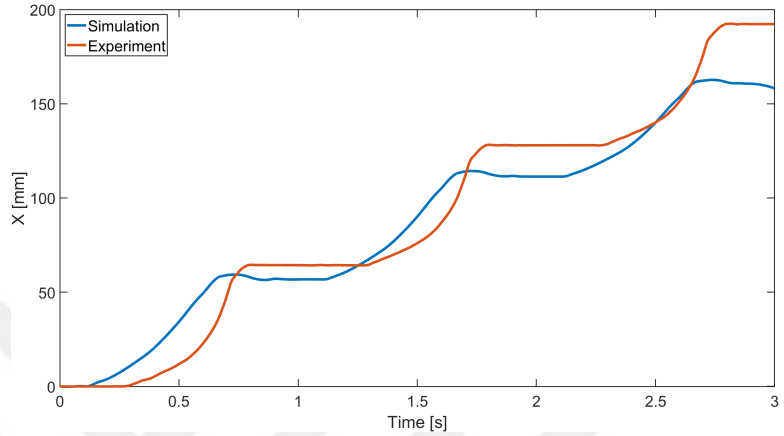
Fig. 3.10(a) shows the position of the module on the x-axis. The blue line shows the simulation result, while the red line shows an experiment. The slight difference is caused because we cannot perfectly match the physical coefficients. Fig. 3.10(b) shows the position of the module on the y-axis. Different from the results in MATLAB, an obvious gliding toward the bottom side is seen. That gliding happens because the parts in the assembly cause a change in the center of mass (CoM) of the module. For instance, the battery, positioned slightly left,

brings the CoM left. Fig. 3.10(c) shows the position of the module on the z-axis.

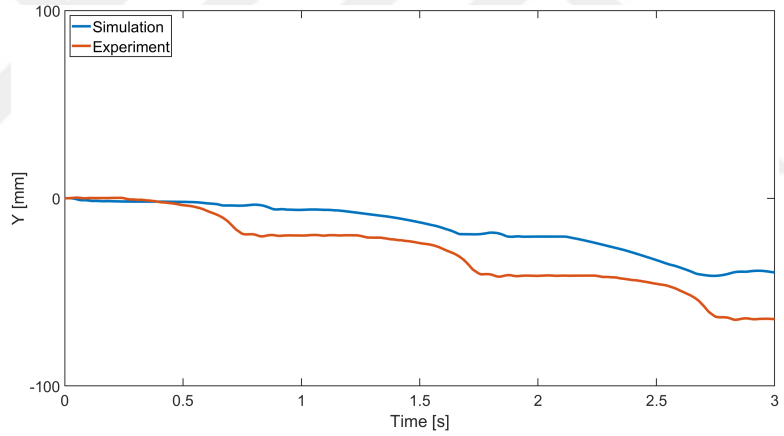
Another beneficial point of using Webots is its ease of importing different robot configurations. One can import robots directly from CAD software. Fig. 3.11(a) shows the ReMBot with three modules, and Fig. 3.11(b) shows the ReMBot with four modules. Also, we could simulate different backbone types. We investigated both the backbones with cylinder magnets and the backbones with square prism magnets. We assumed that there was a rotational joint when backbones with cylinder magnets were used. Because the cylinder magnets separate from each other when modules bend too much in experiments, we restrained the rotational motion between modules in simulations. On the other hand, we assumed that the connection is rigid when backbones with square prism magnets are used. In experiments, both magnet types break up after they try to climb an obstacle they cannot. In simulations, they try to climb but cannot, but the separation never happens. So, we wait 20 *seconds* to see whether the robot can climb the obstacle in simulations.

With all this freedom and capability that Webots gives us, we simulated the obstacle climbing experiments mentioned in Sec. 6. In those experiments, we investigated the effects of the total number the robot consists of, and the backbone types the modules use. In simulations on Webots, we investigated the obstacle climbing capability of two modules, three modules, and four modules moving together. We also looked at the obstacle climbing capability while they are connected with backbones using cylinder magnets and backbones using square prism magnets. We presented a detailed comparison between the simulation and experiment results in Sec. 6.

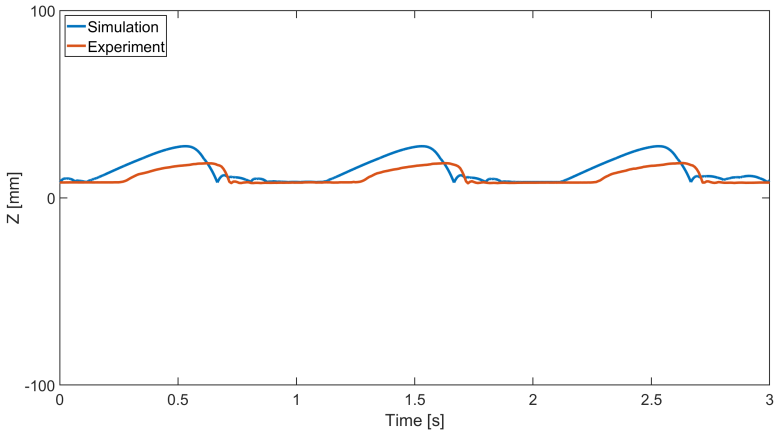
Fig 3.12 shows one example of obstacle climbing simulation conducted on Webots. In this example, two modules are connected to each other by backbones with cylinder magnets. The robot, consisting of two modules, starts to move and climb an 32 mm obstacle. This value in experiments is 27 mm. However, when we consider the perfect conditions the simulation creates, it is expected to climb higher obstacles in simulations.



(a)

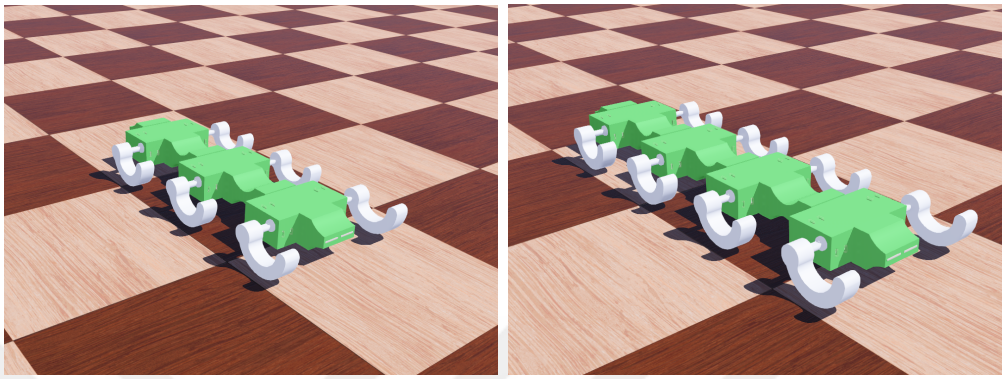


(b)



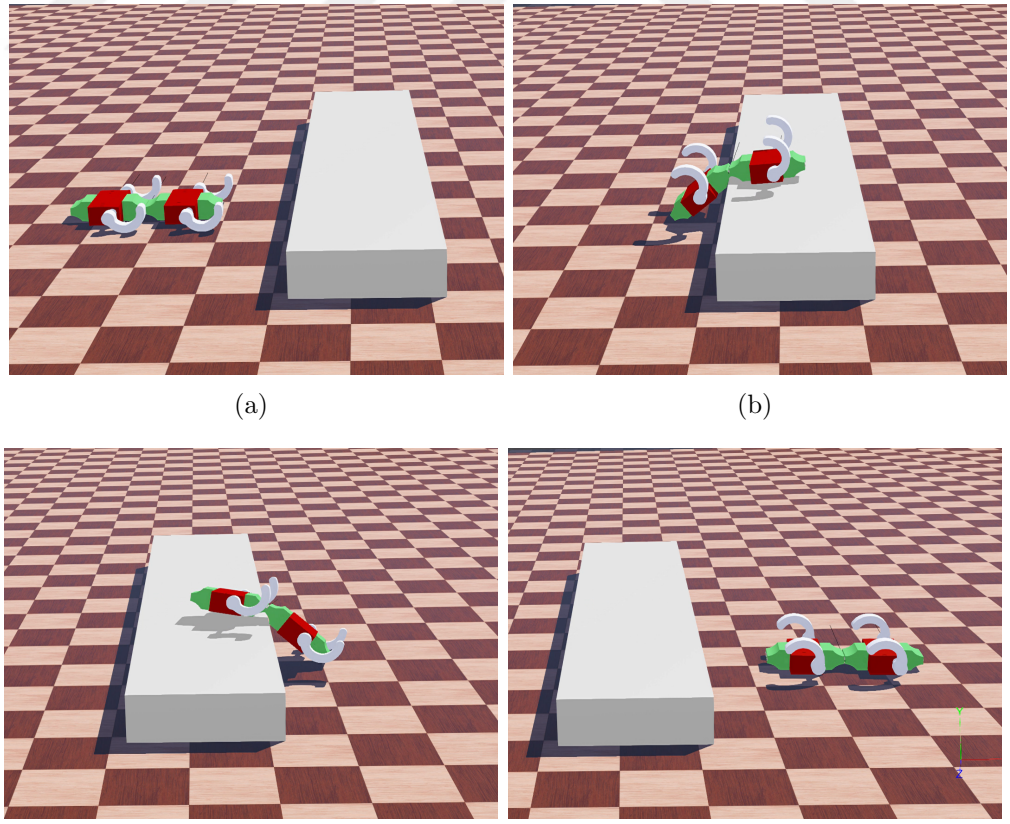
(c)

Figure 3.10: Motion of one module on Webots and experiment.



(a) (b)

Figure 3.11: (a) Three modules and (b) four modules of ReMBot in Webots environment.



(a) (b)

(c) (d)

Figure 3.12: Two modules of ReMBot climb obstacle on Webots.

Chapter 4

Control

In robotics, control means having authority over the physical system you want to regulate. As a basic example, consider a DC motor. Even though we have a full mathematical model, it is not certain that we can have the expected velocity for a given input voltage. This might be simply happening because of manufacturing errors on the motor. In that case, we can measure the motor's rotational velocity using an additional sensor and change the motor's input voltage to get the desired rotational velocity. As described in this example briefly, in robotics systems, we can build different controllers for the varying needs of the system, like positioning the robot accurately, using the manipulator correctly, or steering the robot to the correct place.

There are different control requirements in mobile robots. These controllers are developed for the diverse needs of the robot. For instance, a controller can control the rotational velocities of the wheels in a mobile robot. One needs a rotational encoder to have the motor's rotational velocity as feedback. A more complicated example of the controllers might be controlling the robot's position. For implementing such a controller, the developer could use different sensors for feedback and combine them using Kalman Filter techniques. For example, the LIDAR feedback can be combined with rotational encoder data using Kalman Filters for accurate positioning.

In our system, two different controllers are working to operate modules. The first one is a Linear Quadratic Gaussian(LQG) controller, which controls the position and rotational velocity of the legs of the modules. Gait means the walking pattern of the robot. The gait of the legged robot can be changed by setting phase differences between the legs. The LQG controller helps us to choose different gaits by controlling the position of the leg. The second controller is the linear feedback controller, which controls the position and orientation of the robot. This controller enables our modules to move desired goals correctly. Both the LQG controller and linear feedback controller, with their important tasks, enhance the abilities of our modules.

The linear feedback controller can be listed as the outer controller, while the LQG controller can be listed as the inner controller. The working principles of those controllers are shown in Fig. 4.1. First, the motion capture system passes the position and orientation of the modules as input to the linear feedback controller. Then, it outputs the required rotational velocity values for each leg of the moving module. The LQG controller gets those rotational velocity values as input and outputs the voltage values to DC motors in each leg. This control loop runs until the module gets close enough to the goal position.

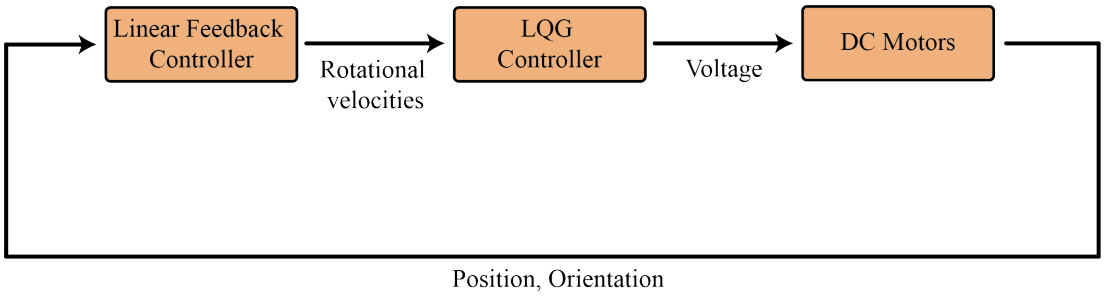


Figure 4.1: Control scheme of the ReMBot.

In addition to these controllers, there is also a synchronization algorithm enabling modules to move together when they are connected. This algorithm, even if it is not a controller, will be investigated in this chapter. In the following part of this chapter, the LQG controller, the linear feedback controller, and the communication algorithm are explained in detail.

4.1 Linear Quadratic Gaussian (LQG) Controller

ReMBot modules can operate on different gaits with the help of position control. Operating legged robots on different gaits requires position and rotational velocity control on the legs. Our robot's palm-size dimensions restrict us from using complicated sensors that can give precise position and velocity data at the same time. In addition, the cost and size limitations force us to use DC motors which are cheap and small. The LQG controller we implemented overcomes issues the sensor and actuator create and enables our modules to run with a defined gait and rotational velocity [15].

The development of the LQG controller starts with the modeling of the DC motor. Using Kirchoffs's Voltage Law (KVL), we can write the voltage equation of the DC motor as follows:

$$v = Ri + L\frac{di}{dt} + \epsilon = Ri + L\frac{di}{dt} + K_e\omega \tag{4.1}$$

where v is the applied voltage, R and L are the armature resistance and inductance, respectively. ϵ is the back emf, which is related to the K_e, the back emf constant of the motor, and i denotes armature current and ω is the rotational velocity of the motor.

We can get the following equation by applying Newton's Second Law:

$$J\frac{d\omega}{dt} = T - b\omega = K_t i - b\omega \tag{4.2}$$

where J and b are the moments of inertia and damping coefficient of the rotor, respectively, T is the toque motor applies, and K_t is the motor torque constant.

The DC motor manufacturers do not often report the physical parameters of their

motors directly but rather give motor specifications in terms of the stall and no-load free-run performance at a rated voltage. These specifications are, namely, free or no-load current, no-load speed, stall current, and stall torque.

Using Equation 4.1 and considering the fact that rotational velocity at stall moment is zero, ($\omega_{\text{stall}} = 0$) and $\frac{di}{dt} = 0$ for a steady state performance, the armature resistance can be found by:

$$R = \frac{v}{i_{\text{stall}}} \tag{4.3}$$

where i_{stall} is the current on stall condition. Likewise, the back emf constant can be found by:

$$K_e = \frac{v - i_{\text{free}}R}{\omega_{\text{free}}} \tag{4.4}$$

where i_{free}, ω_{free} are the current and rotational velocity when the motor turns without a load applied.

The torque constant is determined from stall torque and current, i.e., where ω reaches zero in Equation (4.2):

$$K_t = \frac{T_{\text{stall}}}{i_{\text{stall}}} \tag{4.5}$$

where T_{stall} is the moment exists in stall condition.

The rotor damping is not easily measurable and is not always provided on a manufacturer datasheet. An alternative is to use the no-load current to infer a value for rotor damping. For no-load condition, $\frac{d\omega}{dt} = 0$ will hold and, from Equation (4.2), the generated mechanical torque must equal the rotor damping torque, which yields:

$$b = \frac{i_{\text{free}} K_t}{\omega_{\text{free}}} \tag{4.6}$$

After calculating the parameters for the DC motor, we can define states for the system we are trying to control. For the DC motor we can list three states; namely position (θ), velocity (ω), and current (i), as follows:

$$\boldsymbol{x} = \begin{bmatrix} \theta \\ \omega \\ i \end{bmatrix} \tag{4.7}$$

Equations 4.1 and 4.2 can be rewritten using a state space representation as follows:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u} \\ \boldsymbol{y} &= \mathbf{C}\boldsymbol{x} + \mathbf{D}\boldsymbol{u} \end{aligned} \tag{4.8}$$

where $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{y}$ denote the state, input, and output vectors, respectively, and matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are given by:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \frac{-b}{J} & \frac{K_t}{J} \\ 0 & \frac{-K_e}{L} & \frac{-R}{L} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix}$$
$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix}$$

Note that the system input is motor voltage, $\boldsymbol{u} = \begin{bmatrix} v \end{bmatrix}$, and the only measurable output is the angular position of the DC motor, $\boldsymbol{y} = \begin{bmatrix} \phi \end{bmatrix}$. This is simply because, in our robot, we can only measure the positions of the DC motors using

the rotary position sensors. Since **A**, **B**, **C**, and the number of the states are known, controllability and observability can be checked by looking at the ranks, respectively:

$$rank\left(\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B}\end{bmatrix}\right) = 3 \tag{4.9}$$

$$rank\left(\begin{bmatrix}\mathbf{C} & \mathbf{CA} & \mathbf{CA}^2\end{bmatrix}^\top\right) = 3 \tag{4.10}$$

It can be seen that the system is both observable and controllable, implying full rank [22].

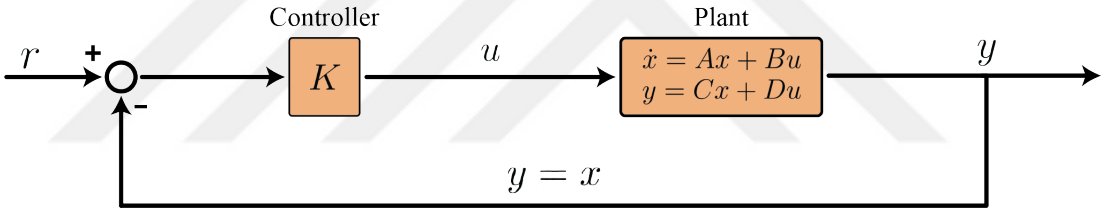


Figure 4.2: Block diagram showing a linear feedback controller.

After modeling the DC motor, we can start the development of the controller. As shown in Fig. 4.2, the controller gain matrix (**K**) must be placed to control the system. This controller takes the output $\boldsymbol{y}$ and multiplies with **K** and feedbacks to the output as following:

$$\boldsymbol{u} = \mathbf{K}(\boldsymbol{r} - \boldsymbol{x}) \tag{4.11}$$

We need knowledge of the full state of the system to use the control law. That means we have to measure all our states directly or estimate those we do not measure. To implement gait adjustment in our robot, we define the desired state at any given time as:

$$\boldsymbol{r} = \begin{bmatrix} \beta_{0j} + 2\pi f_{\mathrm{j}} t \\ 2\pi f_{\mathrm{j}} \\ i_{\mathrm{free}} \end{bmatrix} \tag{4.12}$$

The initial constant phase (β_0), which varies between different legs, is given to maintain the desired gait. The position element is wrapped between $[0, 2\pi)$ radians. Each motor has a unique vector set to describe the desired state. To indicate which motor is controlled, subscript j is used, and the user gives frequency (f) to define the desired velocity.

One can also place closed-loop poles manually to get the desired response. However, pole placement might not give sensible results if the procedure is done carelessly. Thus, Linear Quadratic Regulator (LQR) is designed to place the poles based on error and control effort. LQR's main purpose is to minimize the control effort while providing adequate control of the reference signal. In LQR control, the determination of control gains starts with the minimization of the following cost function:

$$J = \int_0^\infty (\boldsymbol{x}^\top \mathbf{Q} \boldsymbol{x} + \boldsymbol{u}^\top \mathbf{R} \boldsymbol{u}) \, \mathrm{d}t \tag{4.13}$$

in which $\mathbf{Q}$ and $\mathbf{R}$ are weight matrices. Adjustments in these matrices illustrate the relation between state excursion and control effort. This means that an improvement in state tracking cannot be obtained without using more control effort. Also, a reduction in control effort cannot be obtained without sacrificing tracking performance. The minimum of LQR's cost function is found by setting the derivative of the cost function to zero and solving for the control law $\boldsymbol{u}$ [22].

In order to determine weight matrices, $\mathbf{Q}$ and $\mathbf{R}$, Bryson's rule is used. Diagonal elements of the $\mathbf{Q}$ and $\mathbf{R}$ matrices are chosen based on the maximum error of the states:

$$\mathbf{Q} = \begin{bmatrix} \frac{\rho_1}{{x_{1,max}}^2} & 0 & 0 \\ 0 & \frac{\rho_2}{{x_{2,max}}^2} & 0 \\ 0 & 0 & \frac{\rho_3}{{x_{3,max}}^2} \end{bmatrix} \tag{4.14}$$

$$\mathbf{R} = \begin{bmatrix} \frac{1}{{U_{1,max}}^2} \end{bmatrix} \tag{4.15}$$

Weighing factors $\rho_{1,2,3}$ in $\mathbf{Q}$ matrix are selected based on the needs of the robot. Selecting $\mathbf{R}$ as a small value will be helpful to consume less energy, whereas large values will be better for controller performance.

The control gain matrix $\mathbf{K}$ can be defined as follows:

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^{\top}\mathbf{S} \tag{4.16}$$

where $\mathbf{S}$ is obtained by solving the following Algebraic Riccati Equation:

$$\mathbf{A}^{\top}\mathbf{S} + \mathbf{S}\mathbf{A} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\top}\mathbf{S} + \mathbf{Q} = 0 \tag{4.17}$$

MATLAB provides a built-in function that gives the $\mathbf{K}$ matrix when you provide the matrices that represent the physical system which is $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$. So that, after modeling the system, one can try new $\mathbf{Q}$ and $\mathbf{R}$ values and investigate the performance of the controller.

The LQR controller can control the motors successfully. However, to control all states in a state vector ($\boldsymbol{x}$), we need to measure all states or estimate them. Equation 4.10 proves that we can estimate other state variables with only position measurement. We implemented a Linear Quadratic Estimator (LQE), to estimate the state variables and placed it before the LQR controller, as shown in Fig. 4.3.

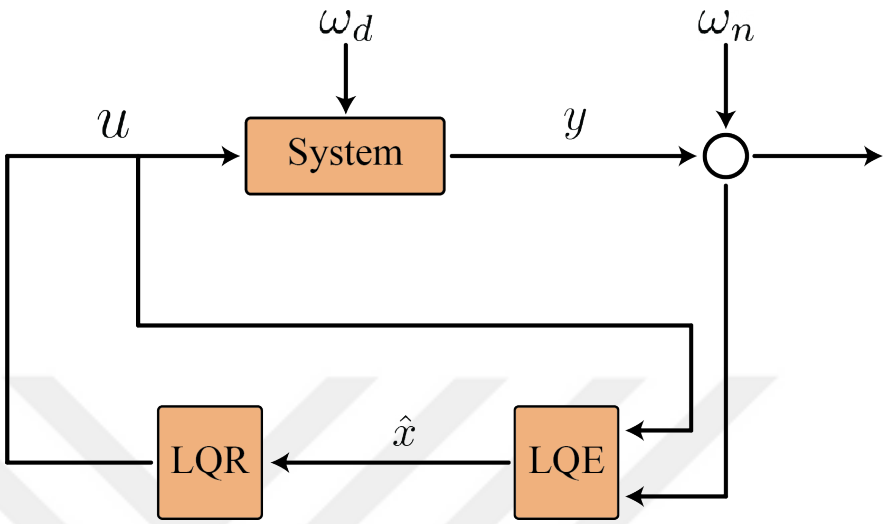


Figure 4.3: Block diagrams showing a Linear Quadratic Gaussian (LQG) controller.

In a real system, both the system itself and the measurement are affected by their surroundings. In the block diagrams shown in Fig. 4.3 $\boldsymbol{\omega_d}$ represents disturbance action on the system while $\boldsymbol{\omega_n}$ stands for the measurement noise. However, if we assume both of them as zero, we can write the state estimation dynamics as follows:

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}} &= \mathbf{A}\hat{\boldsymbol{x}} + \mathbf{B}\boldsymbol{u} + \mathbf{K_f}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \\ \hat{\boldsymbol{y}} &= \mathbf{C}\hat{\boldsymbol{x}} \end{aligned} \tag{4.18}$$

where $\hat{\boldsymbol{x}}$ is the state estimation vector and $\mathbf{K_f}$ is the estimator gain matrix.

After the state estimation vector is introduced, we can add the disturbance ($\boldsymbol{\omega_d}$) and measurement noise ($\boldsymbol{\omega_n}$) as follows:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u} + \boldsymbol{\omega_d} \\ \boldsymbol{y} &= \mathbf{C}\boldsymbol{x} + \boldsymbol{\omega_n} \end{aligned} \tag{4.19}$$

As we did in the LQR case, we have to define weight matrices related to $\boldsymbol{\omega_d}$ and $\boldsymbol{\omega_n}$ so that we can define a cost function and find the estimator gain matrix $\mathbf{K_f}$.

We can define $\mathbf{Q_N}$ related to the $\boldsymbol{\omega_d}$ and $\mathbf{R_N}$ related to the $\boldsymbol{\omega_n}$. In our case $\mathbf{Q_N}$ has a 3x3 dimensions and $\mathbf{R_N}$ has 1x1 dimension. The choice of those values affects the behavior of the estimator. Small $\mathbf{R_N}$ means that the measurement noise is small so that the estimator relies more on the measurement data rather than the model. On the other hand, having large $\mathbf{R_N}$ means that the measurement is noisy, and in this case, the estimator relies more on the model.

The goal of the optimization here is to minimize the error between the estimation and the measurement and find the best possible $\mathbf{K_f}$ as follows:

$$L_{LQE} = \mathbb{E}((\boldsymbol{x} - \hat{\boldsymbol{x}})^\top (\boldsymbol{x} - \hat{\boldsymbol{x}})) \tag{4.20}$$

At the end of this optimization, we can get the $\mathbf{K_f}$, estimator gain matrix :

$$\mathbf{K_f} = \mathbf{P}\mathbf{C}^\top {\mathbf{R_N}}^{-1} \tag{4.21}$$

where $\mathbf{P}$ is obtained by solving the following Algebraic Riccati Equation:

$$\mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^\top - \mathbf{P}\mathbf{C}^\top {\mathbf{R_N}}^{-1}\mathbf{C}\mathbf{P} + \mathbf{Q_N} = 0 \tag{4.22}$$

The LQE and LQR working together constitute the LQG, as shown in Fig. 4.3. In summary, LQE estimates the states and feeds them to the LQR. Then, LQR produces an output and gives it to the physical system, which is, in our case, a DC motor.

In our system, we use DC motors as actuators and rotational potentiometers as sensors. These rotational potentiometers, with their size under a centimeter scale, are an excellent choice for our robot design. However, they have a drawback as well. They have a 'dead zone' where the data cannot be trusted. The dead zone occurs between $[-15°, 15°]$, as can be seen in Fig. 4.4. The characteristics and the dead zone of the sensor must be considered while designing the LQE.

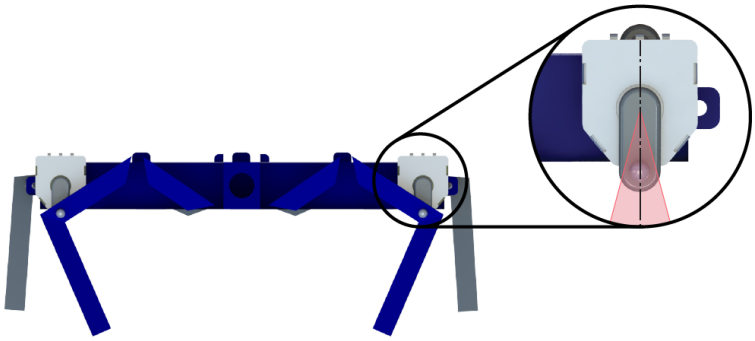


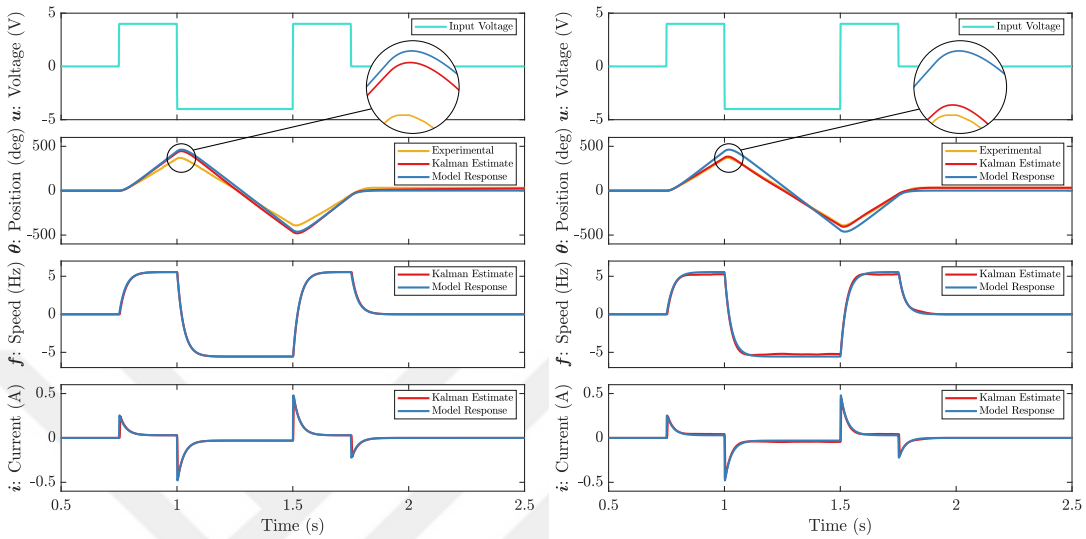
Figure 4.4: Rotary position sensor and its dead zone are shown with a miniature quadruped.

We utilize a 'Hybrid' controller in our robot to tackle the problem that the dead zone creates. The 'Hybrid' controller can modify the trust regions for the model-based estimation, and the sensor data, i.e., the controller in certain regions can prefer to trust the model more, whereas, in other regions, it will prefer to trust the sensor readings more. The main reason why we chose the hybrid controller for the system is the error that occurred in the sensor that is used in the robot.

Sensors have two different zones, which must be investigated separately to estimate measurement noise in the system. To calculate the covariance of the measurement noise in the reliable zone, three motors are actuated at 2 Hz, and one motor is put stationary in the reliable zone. This procedure is repeated for each motor, and the mean of the four values is taken as $\mathbf{R_N}$. The same method is applied to the dead zone by placing one of the motors in the dead zone at rest. From the experiments, we calculated the following $\mathbf{R_N}$ values:

$$R_{\text{N,reliable}} = 5.9 \times 10^{-6}\ rad^2$$
$$R_{\text{N,dead}} = 3.1\ rad^2$$

From the values above, it can be seen that there is a significant difference in the measurement noise, and the dead zone measurements are not reliable. Thus two different sets of values should be calculated for both zones. In the reliable zone,



(a) (b)

Figure 4.5: LQE output (Kalman estimate) of the system for the rotational position, rotational velocity, and current (a) when parameters for the dead zone are used and (b) when parameters for the reliable zone are used.

the LQE will be influenced mainly by measurement and mostly by the system model in the dead zone. To test out our theory, a pre-determined voltage signal is fed to the motors, and their output, as well as model outputs, are recorded. Fig. 4.5 shows the position, rotational velocity, and current responses of the LQE and the model. Also, for the position response, we present experimental results to investigate the success of the estimator. While we utilize parameters for the dead zone ($\mathbf{R}_{\text{N,dead}}$), as in Fig. 4.5(a), the Kalman estimate (red line on the second plots) relies more on the model response. On the other hand, while we utilize parameters for the reliable zone ($\mathbf{R}_{\text{N,reliable}}$), as in Fig. 4.5(b), the Kalman estimate relies more on the experimental data, which means the sensor reading.

After opting to use a hybrid controller, the performance of the controller is experimentally evaluated on the actual robot. In Fig. 4.6, solid black lines represent the reference values for phase and rotational velocity values in the first two plots. The blue lines represent the system response when the hybrid controller is not used, and the red lines represent the results when the hybrid controller is used.

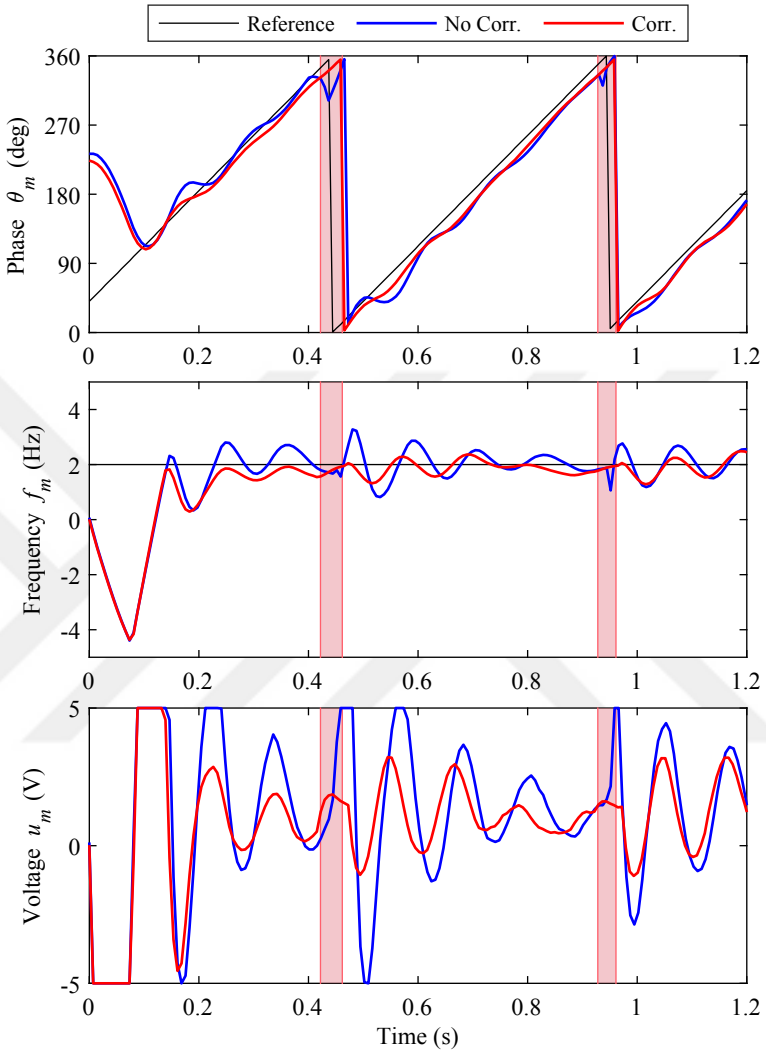


Figure 4.6: Controller performance shown with the reference signals, system response with and without hybrid controller used.

The last plot represents the voltage applied to the motor as the controller's output. The difference in the undulations on the red and blue lines in these plots clearly shows the advantage of a hybrid controller.

With the LQR controller applying inputs and the LQE providing the estimates, our robot becomes ready for operation. The user can select the gait and the rotational velocity of the legs. In the path-tracking operations, this LQG controller takes the inputs of leg phases and rotational velocity values as the outputs of the path-tracking controller.

4.2 Path-Tracking

The ReMBot modules can track a path to reach a given position. This enables them to move to each other for the docking operation, which is the essential operation making our robot a reconfigurable robot. Along with this, path-tracking allows them to move to goals when modules are separated. The path-tracking controller described in [23] is implemented in our system. The input for this controller is calculated according to the data received from the motion capture system then the outputs of the path-tracking controller are transmitted to modules wirelessly.

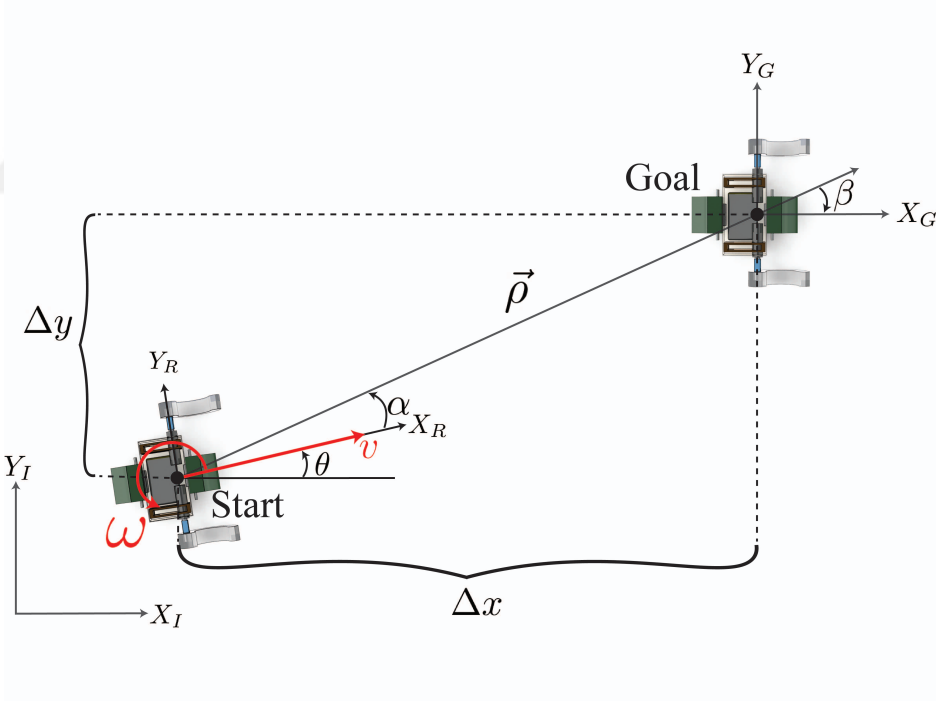


Figure 4.7: Path-tracking modeling of one module moving from the start pose to the goal pose.

The path-tracking problem of the robot can be defined as moving from a start pose to a goal pose, as in Fig. 4.7. This problem can be solved for a differential-drive robot by implementing a feedback controller proposed in [23]. The ρ , α, and β values in Fig. 4.7 can be defined as:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{4.23}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x) \tag{4.24}$$

$$\beta = -\theta - \alpha \tag{4.25}$$

where $X_I - Y_I$ represent the inertial frame, $X_R - Y_R$, represent the robot frame, and $X_G - Y_G$ represent the goal frame. ρ is the distance between the robot frame and the goal frame. α is the angle from $\boldsymbol{X_R}$ to $\boldsymbol{\rho}$, θ is the angle from $\boldsymbol{X_I}$ to $\boldsymbol{X_R}$ and β is the angle from $\boldsymbol{\rho}$ to $\boldsymbol{X_G}$. The motion capture system gives us the positions and orientations of the modules so that we can calculate the values above.

After calculating these values, translational velocity (v) and rotational velocity (ω) of the robot can be assigned as:

$$v = k_\rho \rho \tag{4.26}$$

$$\omega = k_\alpha \alpha + k_\beta \beta \tag{4.27}$$

where k_ρ, k_α and k_β are controller constants. The velocity values found above can be used directly to determine the rotational velocities of the right and left DC motors. However, the translational velocity decreases while the module comes closer to the goal position, resulting in lower rotational velocities for the left and right motors. But our motors are not controllable in such low rotational velocities. To overcome this issue, we carried out the following operations and updated the translational velocity (v) and rotational velocity (ω):

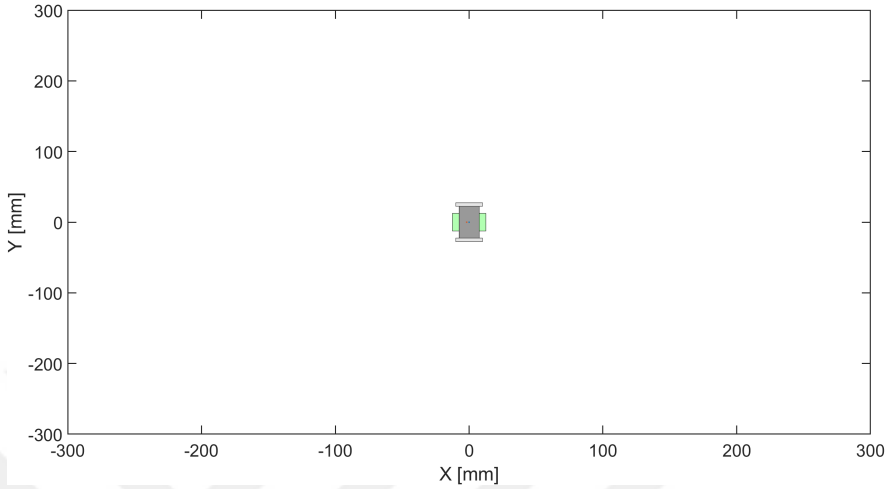


Figure 4.8: Simulation environment developed to tune path-tracking controller parameters.

$$v := \frac{v}{|v|\kappa} \tag{4.28}$$

$$\omega := \frac{\omega}{|v|\kappa} \tag{4.29}$$

where κ is the constant that determines the new translational velocity such that if we increase it the module moves with less translational velocity. After calculating these new velocities, we determined the rotational velocities for the right and left motors as follows:

$$\omega_{\text{right}} = \frac{v + l\omega}{r} \tag{4.30}$$

$$\omega_{\text{left}} = \frac{v - l\omega}{r} \tag{4.31}$$

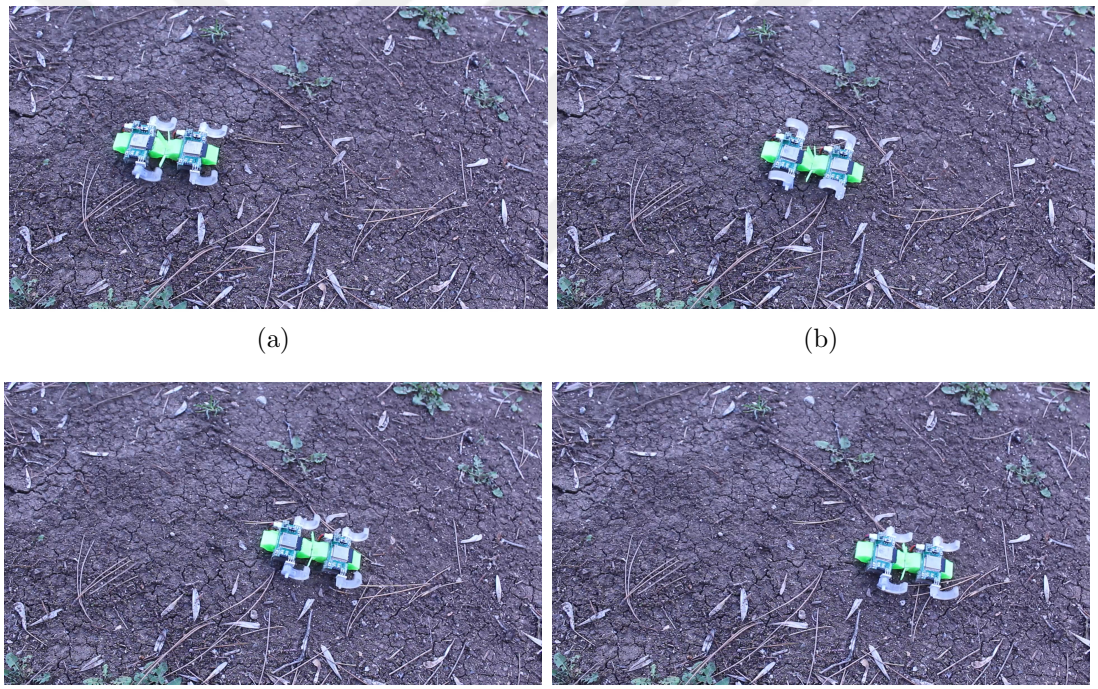
where l is the distance from the center of rotation of the module to feet, and r, is the wheel's radius, which in our case, is 28 mm.

We created a simulation environment on MATLAB, shown in Fig. 4.8, before implementing the path-tracking controller. This environment helped us to tune controller constants k_ρ, k_α, k_β, and κ. While developing the simulation, we assumed our robot was a different-drive standard wheeled robot. Even though we have a dynamic modeling environment, we used the simple differential-drive assumption because its simplicity enables us to try different controller constants more quickly. During the tuning process, we focused on having a smooth path while keeping the outputs in a range so that our DC motors could still run.

4.3 Synchronization Algorithm

The modules, with their individual capabilities, can move around and can complete some tasks solely. However, they must be connected to accomplish some tasks, such as climbing or walking on different terrains. The motions of the legs of the modules must be synchronized when modules are connected. Any irregularity between the leg phases might cause the modules to be unsuccessful in the task, or it might cause them to be separated.

We developed a synchronization algorithm to coordinate the leg movements of modules moving together. The program inside the microcontroller sends the position of the 'leader' module to other modules after the connection is detected. The connection sensors described in Section 2.2.4 inform the microcontroller about the connection. After that command, the path-tracking controller stops working. The 'leader' module starts working on defined rotational velocity on pronk gait which two legs have no phase difference. The leader, then, sends its legs' positions to other modules in each 50 ms. Other modules, take these positions and give them as input values to the LQG controller. This process allows our modules to synchronize and move together properly. Fig. 4.9 shows how modules move together in a soil terrain while the modules toward the right from Fig. 4.9(a) to Fig. 4.9(d).



(a) (b)

(c) (d)

Figure 4.9: Two ReMBot modules moving on a soil terrain.

Chapter 5

Docking and Undocking

The docking and undocking capabilities enable our modules to construct different structures for changing tasks. For instance, if the modules encounter an obstacle they cannot climb single-handedly, the user commands them to dock. After they dock and climb the obstacle, the operator can command them to undock so that the modules can continue their own tasks.

The docking procedure starts with the user's command. Then our system performs the path-tracking algorithm to move one module to another. The magnets in modules attract each other when they are close to each other sufficiently. In this attraction phase, the soft extensions around the backbones help the docking process by eliminating the issues caused by misalignments. After the modules dock, they inform each other with the help of the sensors in magnets. This docking information is then used for syncing the legs and moving the modules together.

The undocking process starts with the user's command as well. After receiving the undocking signal, the module on the front turns its legs backward, while the module on the rear turns its legs forward. In cases where the robot consists of more modules than two, the undocking process also happens similarly. For instance, if a robot consists of three modules and users want to undock the third

module, the first two modules on the front side turn their legs backward while the module on the rear side turns its legs forward.

The use of permanent magnets does not require any additional power and increases the operation time, which is extremely important in an untethered, miniature robot with a limited power source. However, using permanent magnets creates some design problems that must be solved. For example, two permanent magnets connected to each other cannot be separated immediately. The system has to create an appropriate force to separate them. The need for that force creates the most important design problem for backbones. They have to be strong enough to attract each other and stay connected when modules are moving together. Conversely, they must break up when the operator commands modules for undocking. After many trials, we decided to use two cylindric magnets 3mm in diameter and 8mm in length placed on the tip of the backbones, which meets all the requirements mentioned.

Chapter 6

Results

This section describes the experiments we conducted to test different design parameters and choices. During the design, manufacturing, and operation cases, we faced some challenges and had to find the best design parameters to have a better robot. In this section, we describe the definition of 'better' from an engineering perspective with experiments we conducted, and we verify our design parameters are beneficial for the robot.

The experiments we conducted helped us to choose the leg material, and the backbone type, and they can be separated into three main parts, which investigate the followings:

1. The effects of leg compliance on the path-tracking performance
2. The effects of backbone type on docking and undocking performance
3. The effects of module numbers and the backbone type on the obstacle climbing performance

All these experiments discussed in this chapter led us to select soft c-shaped legs and backbones utilizing cylinder-shaped magnets with soft extensions. The soft c-shaped legs performed best in path-tracking experiments compared to rigid

c-shaped legs and standard wheels. The backbones utilizing cylinder-shaped backbones with soft extensions performed the best on docking and undocking experiments. Lastly, we observed that using backbones with cylinder magnets and increasing the module number on the robot are best in obstacle-climbing tasks.

6.1 Path-Tracking Performance

In these experiments, we investigated the effects of leg compliance on the path-tracking performance of our modules. We tried standard wheels, rigid c-shaped legs and soft c-shaped legs. We first determined a simple path-tracking task and compared the simulation and experiment results for each leg type, as shown in Fig. 6.1 to test which type of leg type is more effective for the locomotion of our robot. We look at the suitability by comparing the compatibility of the experiments with simulation results and the smoothness of the path followed by modules.

We did some measurements to quantify this compatibility and smoothness. For compatibility, we looked at the mean value of errors between the experiment and simulation values. For smoothness, we looked at the standard deviation of errors between the experiments and simulation and presented these results in Table 6.1. We repeated the experiments 20 times for each case shown in Fig. 6.1 on a flat table. However, we only showed five trials for each case to have a better representation.

Table 6.1: Mean and standard deviation of error values between the experiments and simulation.

Leg Type	Mean Error (cm)	Standard Deviation (cm)
Wheel	1.31	0.86
Rigid, c-shaped leg	15.83	11.18
Soft, c-shaped leg	1.72	1.05

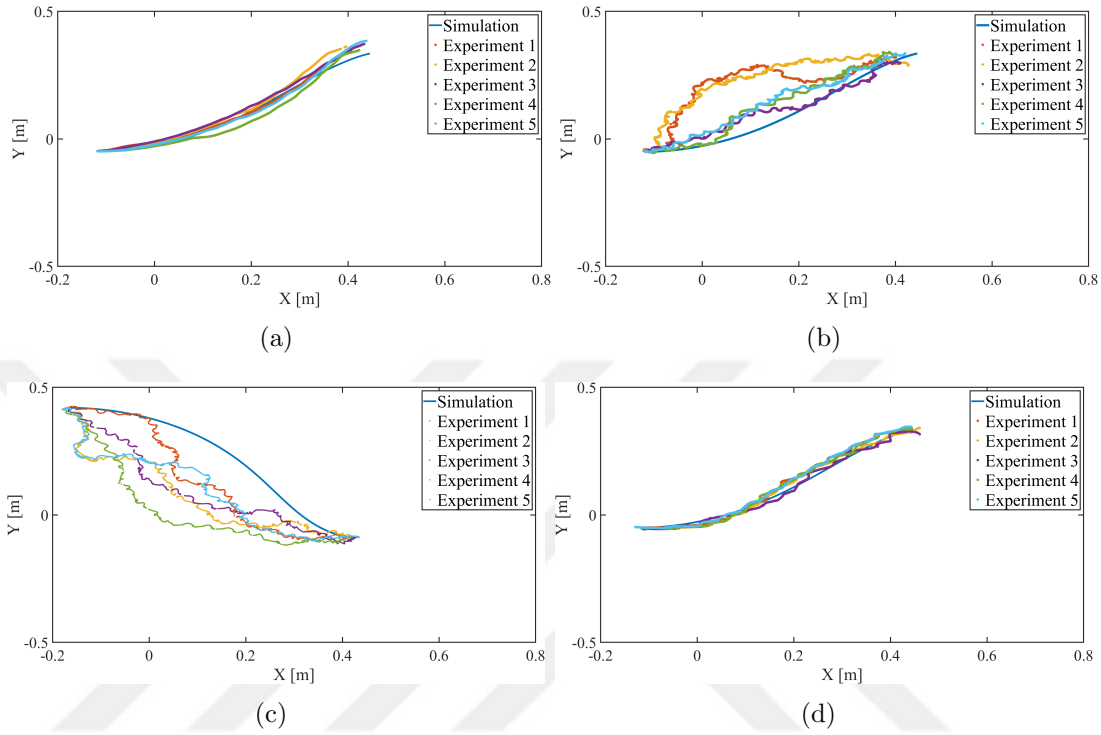


Figure 6.1: Simulation and experiment results of path-tracking of one module with (a) standard wheels, (b) rigid, c-shaped legs (first quadrant), (c) rigid, c-shaped legs (fourth quadrant), (d) soft c-shaped legs.

Fig. 6.1(a) shows the simulation and experiment result for a path-tracking task in the first quadrant. In this experiment, we used a module that uses standard wheels. The blue line in Fig. 6.1(a) represents the simulation result, while the markers in different colors represent experiments. As the differential-drive model is developed for wheeled mobile robots, using wheels instead of legs gives the best results in terms of compatibility between the experiments and the simulation with the mean error of 1.31 *cm*, while using wheels also gives the best results in terms of smoothness with the standard deviation of errors 0.86 *cm* as it could be seen in Table 6.1.

Fig. 6.1(b) and Fig. 6.1(c) show the simulation and experiment results for path-tracking tasks for a module with rigid, c-shaped legs. A clear difference between the simulation and experiments can be seen. This difference is mainly because of the impact created when the c-shaped leg hits the ground. To support this claim and make sure that it is not caused by manufacturing errors, we conducted the

experiments in two different quadrants. Fig. 6.1(b) shows the robot moving in the first quadrant, so in the initial steps, the robot tries to go to the left, but the impact between the right leg and the ground moves the robot too much to the left side. In Fig. 6.1(c), the robot moves in the fourth quadrant, and in the first steps, the impact of the left leg pushes the robot too much to the right side. Other than the difference between the simulation and the experiments, using rigid, c-shaped legs caused a wavy motion of the robot because of the slip between the contact point of the leg and the surface. As a result of the impact and slip, using rigid c-shaped legs give the least successful results with 15.83 *cm* mean error and 11.18 *cm* standard deviation in errors.

Fig. 6.1(d) presents the simulation and experiment results for a path-tracking task in the first quadrant for the robot with soft, c-shaped legs. Using soft material significantly reduces the impact and slip problems that occur in the rigid, c-shaped legs case. Without these problems, soft c-shaped legs give nearly the same results as the wheels. They have a 1.72 *cm* mean error with 1.05 *cm* standard deviation in error, as can be seen in Table 6.1. This result means that when compliant c-shaped legs are used in miniature robots, the modeling and control complexity reduces because we can utilize simple differential-drive kinematics.

Table 6.2: Obstacle heights that one module can climb with different leg configurations.

Leg Type	Obstacle Height (mm)
Wheel	0.9
Rigid, c-shaped leg	7
Soft, c-shaped leg	9

We also conducted another experiment to test the obstacle-climbing capability of one module with different legs. Table 6.2 shows the obstacle heights that one module can climb. In these experiments, soft c-shaped legs performed better than rigid c-shaped legs and wheels because of the friction and the compliance between the leg and the ground.

6.2 Docking and Undocking Performance

In these experiments, we investigated the effects of backbone type on the docking and undocking performance of the modules. The docking and undocking performance is another crucial criterion affecting our robot's overall performance. The attraction area that permanent magnets create helps the module to dock each other. On the other hand, the attraction force also enables the modules to undock from each other by not being too much.

Table 6.3 shows the success rate of docking and undocking experiments. We conducted 20 experiments for each backbone type, both for docking and undocking. We first tried backbones with square prism magnets and backbones with cylinder magnets. Even though cylinder magnets performed better, 55% was insufficient to present our robot as a successfully reconfigurable robot. The bigger c-shaped legs caused our modules to move too much in the vertical direction, leading to misalignment issues between modules. Then, we solved this issue by adding soft extensions to the sides of the backbones with cylinder magnets, as shown in Fig. 2.7. With these soft extensions, the success rate of docking operations increased to 80% in backbones with cylinder magnets and 55% in backbones with square prism magnets. The results prove that the soft extensions do not harm the undocking process, while they greatly improve the docking operations. We can say that by looking at all of these, using backbones with cylinder-shaped magnets and soft extensions is beneficial for our robot.

Table 6.3: Docking and undocking performance of modules with different backbone configurations.

Backbone Type	Docking (%)	Undocking (%)
Square Prism	35	100
Square Prism with Extensions	55	100
Cylinder	55	100
Cylinder with Extensions	80	100

6.3 Obstacle Climbing Performance

In these experiments, we investigated the effects of the total module number constituting the robot and the backbone types on the obstacle-climbing capability of the robot. Our modules, with their c-shaped legs, can climb obstacles by themself. On the obstacle climbing capability, the soft c-shaped legs provide the highest performances. The modules can climb obstacles up to 9 mm with soft c-shaped legs and 7 mm with rigid c-shaped legs. Conversely, when standard wheels are used, they can only climb 0.9 mm obstacles as mentioned in Section 6.1 and in [24].

The reconfigurable nature of our robot allows it to climb higher obstacles than one module cannot normally. After seeing that soft c-shaped legs provide better climbing performance, we decided to use them in our obstacle-climbing experiments. We looked at the effects of different backbones and the total module number on the obstacle-climbing capability of the ReMBot.

We used two different backbone types in our experiments and our simulations. We used backbones with two cylinder-shaped permanent magnets (3 mm in diameter and 8 mm in length, and backbones with two square prism-shaped permanent magnets (3 mm in each side and 1.5 mm in height. The backbones with cylinder magnets provide additional freedom between modules. However, the backbones with square prism magnets act like a rigid connection when magnets are connected.

Table 6.4: Obstacle climbing performance of modules with different backbone configurations and different module numbers.

Backbone Type	2-Modules	3-Modules	4-Modules
Square Prism - Experiments	24 mm	25 mm	29 mm
Square Prism - Simulations	25 mm	26 mm	30 mm
Cylinder - Experiments	27 mm	34 mm	37 mm
Cylinder - Simulations	29 mm	37 mm	39 mm

Table 6.4 shows the height of obstacles in mm that our robot can climb with a different number of modules with different backbone types both in simulation and experiments. The obstacle height increases as the number of modules increases, regardless of the backbone type in simulations and experiments. Likewise, the robots using backbones with cylinder magnets can climb higher obstacles than those using cube prism magnets, regardless of the total module numbers. While the robot configuration with four modules and cylinder magnets could climb a 37 mm obstacle, the robot configuration with four modules and square prism magnets could only climb 29 mm. This difference is due to the additional rotation the cylinder magnets added to the robot. The rotation between modules prevents modules from being unconnected suddenly and holds them together even if they bend too much. However, the square prism backbones act like a rigid backbone of normal walking, and when they face a resisting force, they are suddenly unconnected.

Several reasons generate the differences between the simulation and experiments. The first one is that we simulate the soft legs as rigid, and this causes a difference. Also, because we do not have the perfect physical constants of all surfaces, we conducted experiments which was one other reason we have a difference. Lastly, we modeled the cylinder magnets as rotational joints and square-prism magnets as rigid connections. However, in reality, they do not perfectly act like rotational joints or rigid connections. This discrepancy between the joints' behaviors in simulations and experiments was the last reason. Even though there are differences in results, both the simulations and experiments prove that using backbones with cylinder magnets is more beneficial for our design, so we selected them for our final design.

Chapter 7

Case Studies

We conducted three different case studies showing the abilities of the robot. These examples show the modules' individual capabilities and the robot's capabilities consisting of several modules. They prove our systems' success in path-tracking, docking, and undocking, following sequential orders and synchronous movement of modules.

In our first case study, we investigated the obstacle-climbing capability of modules with different legs, while modules conduct sequential path-tracking duties. Fig. 7.1(a) shows this experiment, the module goes to the goal points sequentially, and while doing this, it climbs an obstacle. In this experiment, we analyzed the maximum heights that one module can climb when it uses different types of legs.

Table 6.2 shows the maximum obstacle height that one module can climb with different configurations. We conducted climbing experiments, as seen in Fig. 7.1(a). The module starts from a point and goes to the first goal point, then moves to the second goal point. And it completes the path by reaching the final goal, as seen in Fig. 7.1(a). We placed a rigid obstacle between the first goal point and the second goal point and observed the maximum height that the module could climb. As it can be seen from Table 6.2, the module with soft, c-shaped legs could climb an obstacle with a height of 9 mm.

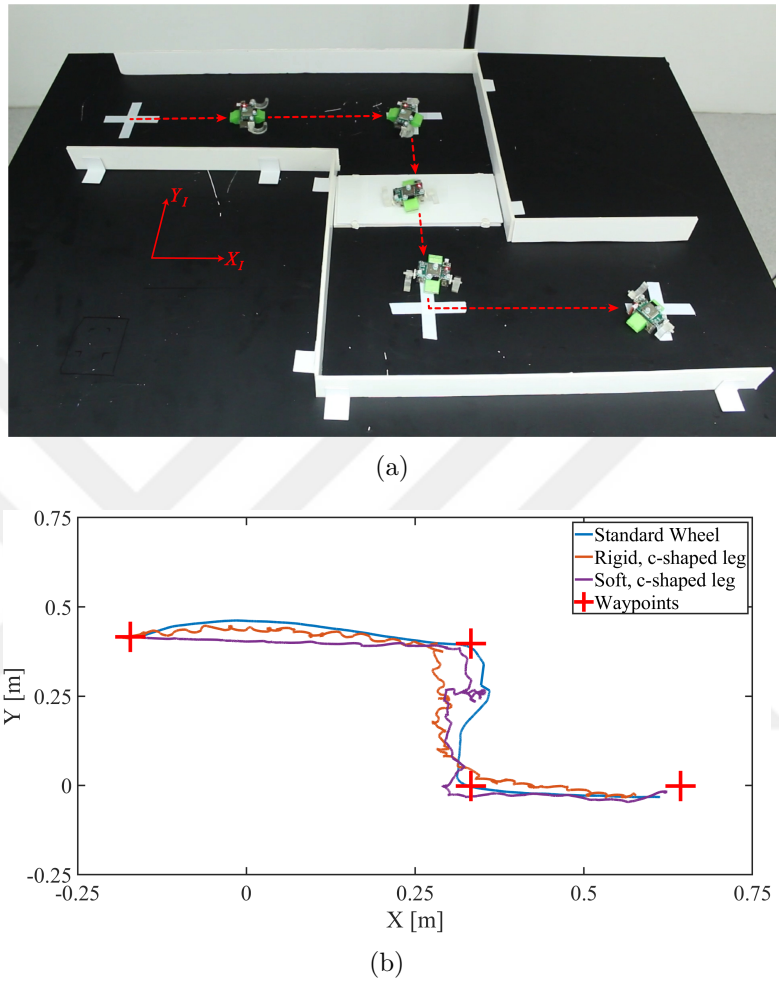
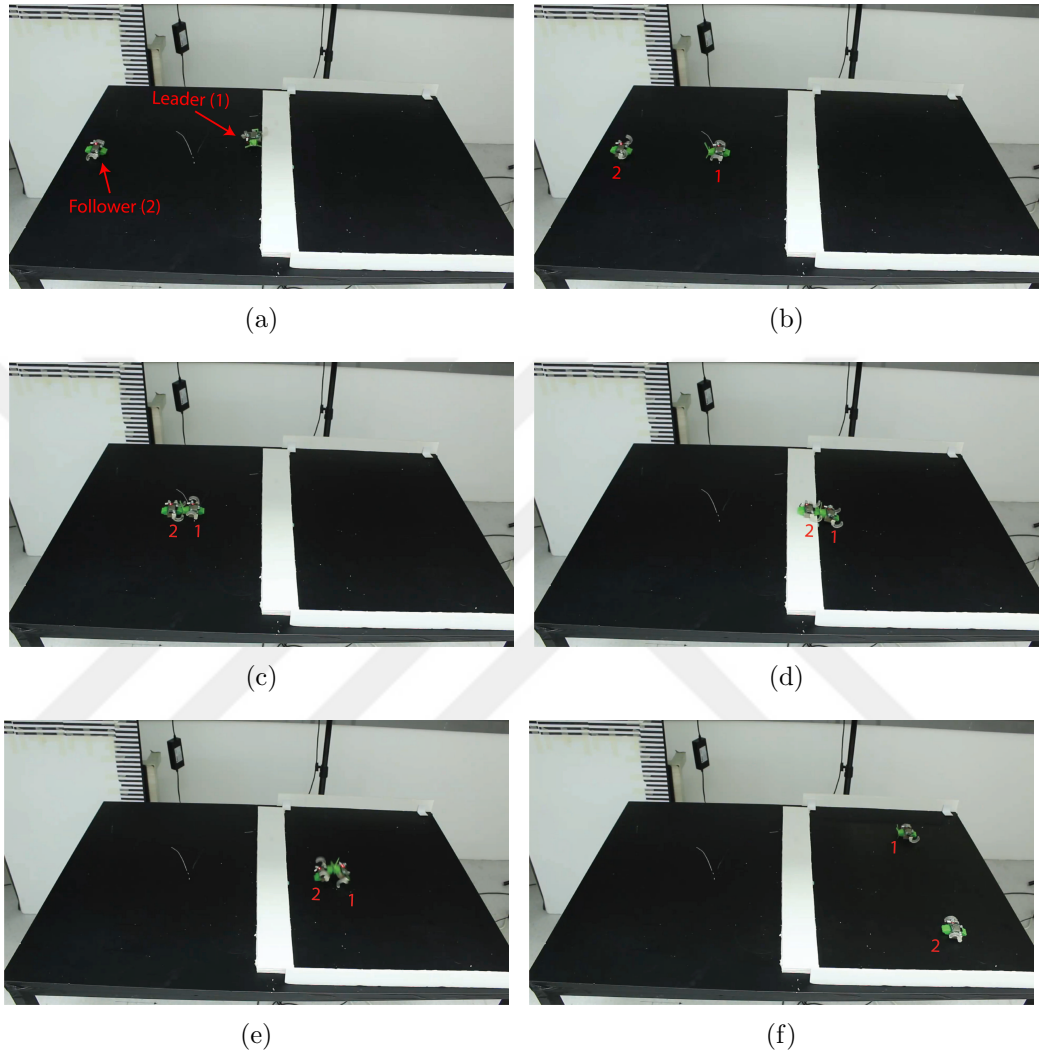


Figure 7.1: A case study that shows the obstacle climbing and path-tracking capability of one module.

Fig. 7.1(b) shows the position of the module while conducting the experiment with different leg configurations. The red plus signs show the goal positions in the sequential tasks. The red, orange, and purple lines represent the position of the module using standard wheels, rigid c-shaped legs, and soft c-shaped legs, respectively. Because the robot with c-shaped legs tries to climb the obstacle in the middle, the plots show oscillations in the position in the middle regions. This experiment showing the advance of the robot in climbing also shows its effectiveness of it in complex missions, e.g., the represented sequential path-tracking experiment such as surveillance missions on different fields.



(a) (b) (c) (d) (e) (f)

Figure 7.2: A case study demonstrates the reconfigurable use of ReMBot with two modules.

We planned two case studies to demonstrate that our robot can work as a reconfigurable robot with all the skills presented in the thesis. These case studies prove that our robots can communicate with each other, conduct path-tracking duties, sense connection, dock, and undock. The software which determines the motions of our robot is written in a modular way so that one can easily implement a new sorted order for the motions of the modules. This simplicity enables us to execute different scenarios with ReMBot modules.

The case study shown in Fig. 7.2 presents the reconfigurable capability of ReM-Bot consisting of two modules. We call the module moving first as *leader*, (shown with (1) in Fig. 7.2), and second as *follower* (shown with (12) in Fig. 7.2) for convenience. The *leader* moves toward an obstacle (20 mm height) that one module cannot climb normally (Fig. 7.2(a)). After trying to climb it user sends a command to abandon the climbing duty. The *leader* returns to the previously defined position by running the path-tracking algorithm and orients itself for docking (Fig. 7.2(b)). Then, the *follower* executes the path-tracking and docks to the *leader* (Fig. 7.2(c)). The sensor on the *leader* detects the connection and reports the server and the *follower*. After sensing the connection, the modules synchronize their legs and move together, which enables them to climb over obstacles (Fig. 7.2(d)). Then, the user sends a command to undock. They undock by using their legs' rotation (Fig. 7.2(e)) and move to separate goal points by running the path-tracking algorithm again (Fig. 7.2(f)). In these motions, user commands and path-tracking outputs are produced in the main computer and then passed to the server. After that, the server transmits the data to modules wirelessly.

In our last case study, we increased the total number of modules to show the scalability of our system, as shown in Fig. 7.3. We numbered each module to follow the figure. In the beginning, two modules (shown as (1) and (2) ,in Fig. 7.3) connected to each other (Fig. 7.3(a)) start to move and go over a 10 mm obstacle (Fig. 7.3(b)). Then, they face a 29 mm obstacle and cannot climb it (Fig. 7.3(c)). After that, the user sends a command to the third module to connect with the first two modules. The third module comes and docks (Fig. 7.3(d)). Then, the sensor in the second module detects the connection and informs the server and other modules about it. Thereupon, three modules synchronize their legs and go over the obstacle (Fig. 7.3(e)). Then the user sends a command to stop the operation of the modules (Fig. 7.3(f)). This case study presents a good example of how we can modify the software to have a robot operating with several modules.

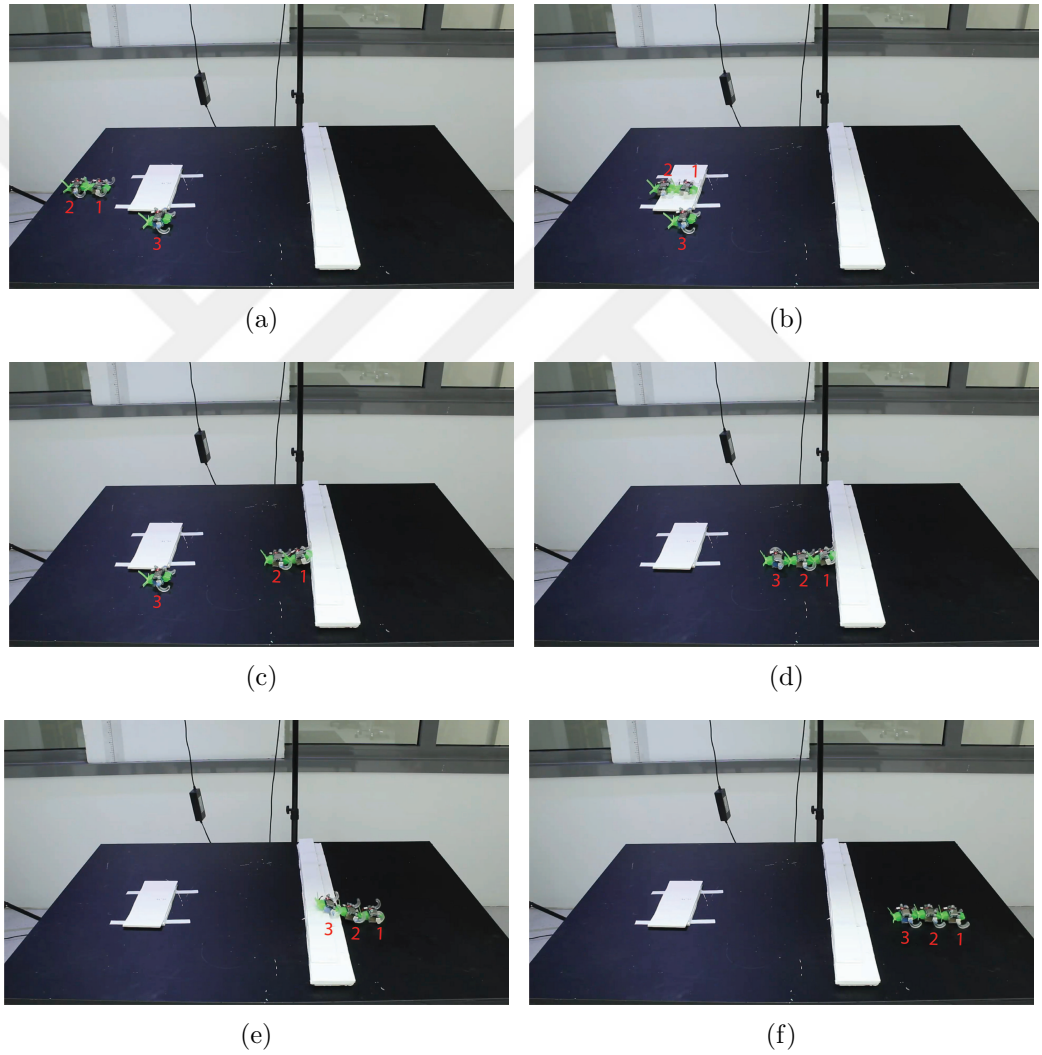


Figure 7.3: A case study demonstrates the reconfigurable use of ReMBot with three modules.

Chapter 8

Conclusion and Future Works

In this thesis, the design of ReMBot, a reconfigurable, miniature, modular robot with a soft connection mechanism, is presented. This robot consisting of modules is capable of wireless communication, path-tracking, docking/undocking, and sensing the connection. Each weighs 29.43 $grams$, while they do not exceed palm sizes. Combining miniature size with operational talents enables ReMBot to climb obstacles in the field and run over different terrains. These miniature-sized robots, along with being cheap and talented, one experienced person can manufacture the modules in around two hours. This ease of manufacturing makes them a great choice for using them in discriminating tasks. One can easily manufacture them and add them to the existing system to get better results on the task.

The existing reconfigurable robots use different methods, such as hook-like active mechanical structures, passive connection-based mechanical structures, or electro-permanent magnets. Even though those robots succeeded in challenging problems, they all need improvements, especially in their connection mechanisms. For instance, the hook-like mechanism needs an additional actuator which complicates the mechanical design and control. On the other hand, the passive mechanical structure creates a rigid connection between modules, preventing them from climbing obstacles or operating on different terrains. Likewise, while creating a rigid connection and constraining the motion, the electro-permanent magnets

also need a complex driving circuit that complicates the control and increases the power consumption. However, our robot using cylinder-shaped permanent magnets overcomes all of these issues. They do not need a driving circuit, do not need an actuator, or do not even consume power at all. In addition to all advantages, the magnets create an additional degree of freedom between modules and help them to climb obstacles and move in different terrains.

With all these design features, our robot, with its soft c-shaped legs, can conduct path-tracking duties successfully. With soft c-shaped legs, our robot can climb 37 mm obstacles, despite its 14.7 mm height, when four modules are connected with backbones with cylinder magnets. Also, with the same backbone configuration, they can perform docking operations with an 80% success rate along with a 100% success rate in undocking operations which is the essential task to being a reconfigurable robot.

As our future works, we plan to add a camera on top of the modules. Then we will implement a vision-based path-tracking algorithm to remove the motion capture system for position feedback. With the help of this camera system, our modules will be able to conduct path-tacking duties in the field without needing a motion capture system. Also, we will improve our dynamic simulations to optimize the design process for a given task so that we can scale up or scale down our module designs and select permanent magnets accurately. We also think that completing path-tracking tasks in different rotational velocity ranges will change the robot's performance. Because of this, we plan to implement an optimization procedure where we optimize the operation of our robot for different rotational velocity ranges and test those ranges with experiments. Lastly, we plan to use variable stiffness soft materials for the c-shaped legs to make our modules more suitable for outdoor operation.

Bibliography

[1] C. Anderson, G. Theraulaz, and J.-L. Deneubourg, "Self-assemblages in insect societies," *Insectes sociaux*, vol. 49, pp. 99–110, 2002.

[2] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[3] T. Tosun, J. Davey, C. Liu, and M. Yim, "Design and characterization of the ep-face connector," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 45–51, 2016.

[4] M. Malley, B. Haghighat, L. Houel, and R. Nagpal, "Eciton robotica: Design and algorithms for an adaptive self-assembling soft robot collective," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4565–4571, 2020.

[5] G. Liang, H. Luo, M. Li, H. Qian, and T. L. Lam, "Freebot: A freeform modular self-reconfigurable robot with arbitrary connection point - design and implementation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6506–6513, 2020.

[6] P. Swissler and M. Rubenstein, "Fireant3d: a 3d self-climbing robot towards non-latticed robotic self-assembly," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3340–3347, 2020.

[7] S. Yi, Z. Temel, and K. Sycara, "Puzzlebots: Physical coupling of robot swarms," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8742–8748, 2021.

[8] S. Niu, L. Ye, H. Liu, B. Liang, and Z. Jin, "Ant3dbot: a modular self-reconfigurable robot with multiple configurations," in *Intelligent Robotics and Applications: 15th International Conference, ICIRA 2022, Harbin, China, August 1–3, 2022, Proceedings, Part I*, pp. 552–563, Springer, 2022.

[9] A. Spröwitz, S. Pouya, S. Bonardi, J. Van Den Kieboom, R. Möckel, A. Billard, P. Dillenbourg, and A. Jan Ijspeert, "Roombots: Reconfigurable robots for adaptive furniture," *IEEE Computational Intelligence Magazine*, vol. 5, no. 3, pp. 20–32, 2010.

[10] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, "Distributed self-reconfiguration of m-tran iii modular robotic system," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 373–386, 2008.

[11] B. Haghighat, E. Droz, and A. Martinoli, "Lily: A miniature floating robotic platform for programmable stochastic self-assembly," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1941–1948, 2015.

[12] Y. Ozkan-Aydin and D. I. Goldman, "Self-reconfigurable multilegged robot swarms collectively accomplish challenging terradynamic tasks," *Science Robotics*, vol. 6, no. 56, p. eabf1628, 2021.

[13] G. Knizhnik and M. Yim, "Docking and undocking a modular underactuated oscillating swimming robot," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6754–6760, 2021.

[14] P. Swissler and M. Rubenstein, "Fireant: A modular robot with full-body continuous docks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6812–6817, 2018.

[15] M. Askari, M. Ugur, N. Mahkam, A. Yeldan, and O. Ozcan, "Control and study of bio-inspired quadrupedal gaits on an underactuated miniature robot," *Bioinspiration Biomimetics*, vol. 18, p. 026004, jan 2023.

[16] M. Askari and O. Özcan, "Dynamic modeling and gait analysis for miniature robots in the absence of foot placement control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9754–9760, 2019.

[17] N. Mahkam and O. Özcan, "A framework for dynamic modeling of legged modular miniature robots with soft backbones," *Robotics and Autonomous Systems*, vol. 144, p. 103841, 2021.

[18] J. L. Meriam, L. G. Kraige, and J. N. Bolton, *Engineering mechanics: dynamics*. John Wiley & Sons, 2020.

[19] A. Vina and A. Barrientos, "C-legged hexapod robot design guidelines based on energy analysis," *Applied Sciences*, vol. 11, no. 6, p. 2513, 2021.

[20] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots," in *2007 IEEE/RSJ international conference on intelligent robots and systems*, pp. 1222–1227, IEEE, 2007.

[21] M. Askari and O. Özcan, "Dynamic modeling and gait analysis for miniature robots in the absence of foot placement control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9754–9760, 2019.

[22] A. E. Bryson and Y.-C. Ho, *Applied optimal control: optimization, estimation, and control*. Routledge, 2018.

[23] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[24] M. Ugur, B. Arslan, A. Ozzeybek, and O. Özcan, "Effects of compliance on path-tracking performance of a miniature robot," in *2023 IEEE 6th International Conference on Soft Robotics (RoboSoft)*, 2023.