FINAL REPORT

Romio and Julibot

Team 11

Theodore Winter | (Leona) Nhi Nguyen | Tuomas Pyorre (Tom)

"We, team 11, ensure that all the work done for the final project is equally divided for each member in the team. Each member has contributed great work to create this play."

I. INTRODUCTION AND OVERVIEW

Due to the impact of COVID19, WPI Drama/Theatre shows are in need of actors due to the restrictions on public performances. And we, gang 11, will bring back everyone's favorite Shakesphere's play : *Romio and Julibot*. With our three actors: Leona's Romi as Romio, Tom's Romi as Tybot and Julibot, and Theodore's Romi as Mercutibot and Fribot.

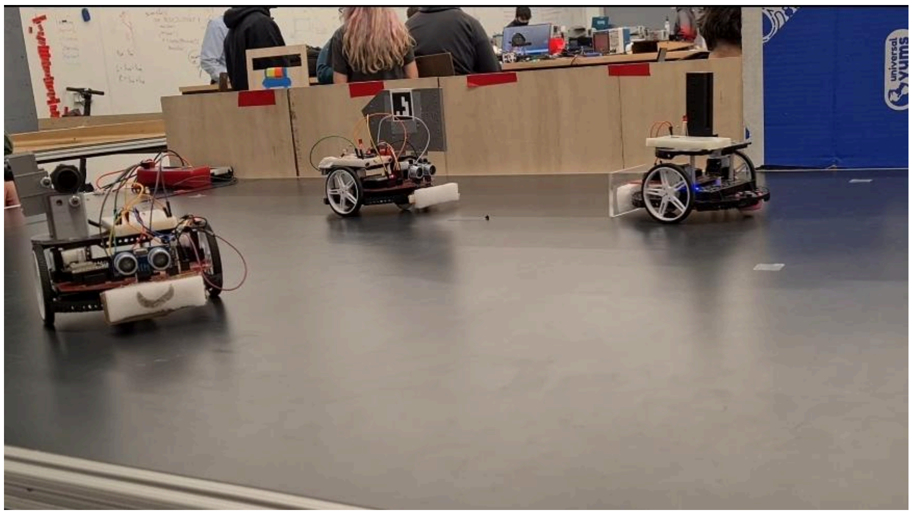


Image 1: A picture from the fight scene with all our romi actors

To recreate the scenes from the play, we have implemented some main functions for our actors: Tybot and Mercutibot must be "circling" each other, Tybot can detect the "stab", Romio is able to find Tybot, and Julibot goes on the ramp stops at the top and leads Romio to follow her as she goes down. All of the main functions will be performed in two scenes: the fight scene and the balcony scene. We will explain more details about the solutions later.

We decided to make MQTT the main communication method between the romis. We also decided to implement all the code in one program with different MQTTState's for different romis. This makes uploading code to each Romi easy, as we only have to change one state. When all code is in one folder, we can easily modify code that affects all Romi's equally, like the MQTT. This also has the benefit of being able to publish the code to Github with relative ease. For example, for Tom's Romi we will upload the robotMQTTState as TYBOT_AND_JULIBOT. The default robotState will always be ROBOT_IDLE. When the scene begins, depending on the button pressed, Tom's Romi will become Tybot or Julibot dependent on the button press and that Romi will tell the other Romis to perform the scene.

II. SOLUTIONS AND JUSTIFICATIONS

For each scene there are main functions, and each function has many options, as well as constraints and challenges. This section only covers the methods and consideration for our team during the play The very details of how we use the methods can be read in the appendices section.

1) Romi Communication

One of the most important challenges to overcome was to devise a means for our robots to communicate with each other. The two primary methods to accomplish this were using the ESP32's Wi-Fi antenna to send messages or IR transmitters and receivers. For our final project we ended up using Wi-Fi for most communication. Although the Wi-Fi was more complicated to set up, it worked regardless of alignment, meaning that there was no possibility of our IR signals being blocked. It also worked from across the room, so we could send signals to the other side of the table with no hassle. There was also less risk of interference from other teams in the lab as the IR receiver has no protection against receiving stray signals from other teams working in the lab. Although the MQTT is not perfect, and can also be suspect to interference if the system is under heavy load. When the room was full of devices, the MQTT could be in constant heavy load, and suddenly break connection. The IR receiver in this sense could be more reliable, it would work as long as we didn't receive IR messages from other teams.

Still, the MQTT is a more powerful communication method, being able to transmit and receive data from multiple different sources simultaneously. Meanwhile the IR receiver cannot receive data from multiple Romi's at the same time, without rigorous planning for the design of each Romi. With IR, we are also not able to transmit and receive signals across the table, while having reliability. So we chose the MQTT as a more reliable communication method between the Romi's. Then, we decided to use the Wi-Fi for a lot of the communication in our demo. To accomplish this, we gave every robot state a corresponding assigned byte. The robots then publish their current state byte to MQTT constantly. The other robots are always monitoring the states of the other two actors and change their own states accordingly.

Our other forms of communication were the IMU collision detection, as well as using two different types of cameras. We needed these too, as although MQTT is great, we also can't continuously only use that to know where the other Romi's were. That's what led us to using the IMU collision detection in a few places, mainly when we wanted the Romi's to know that they were touching each other. For example, when we wanted Tybot to stab Mercutibot, we could send a signal using MQTT that it's their time to stab, and they should stab, but we didn't know if they actually touched. We needed something to tell us if they did, so the IMU collision was an excellent choice in this scenario. Then, when we needed Romio to follow Julibot, we needed the apriltag and the OpenMV camera for Romio to see where Julibot is, and follow them well. Finally, when Romio needed to find Tybot for stabbing, he also needed to know where Tybot was. So, the IR camera and the IR emitter were great choices, as when we got the signal, we could easily go forwards to that direction, and use the IMU collision detection to see when they touched.

Overall, we used a variety of types of communication between our Romi's. Mainly MQTT, unless we needed the Romi's to physically know where each one was, or if we needed something physical to happen between them.

2) <u>The Fight scene</u>

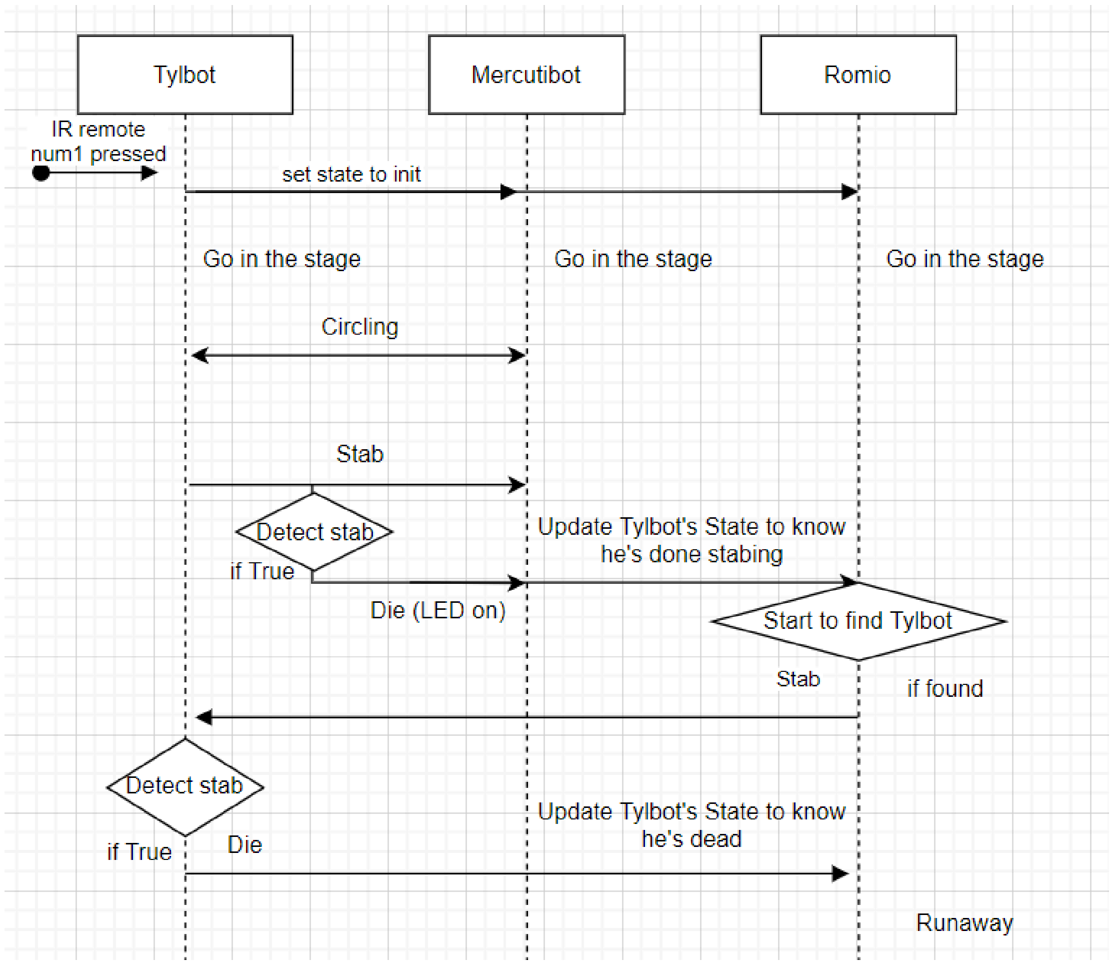


Diagram 1: Fight scene motion diagram

First classic scene in the play where Tybot and Mercutibot dies. For stage entrance in the scene, we all agreed to use ICC kinematics and set the destination point to (40, 0) for all Romis.

❖ a) *The circling act*

This is the scene where Tybot and Mercutibot meet on the stage and start to circle each other before the stab scene. Romio also enters the stage but "hides" in a corner, waiting for his opportunity.

i) Functionality

Tybot and Mercutibot have to circle each other, while Romi is watching. Tybot and Mercutibot will depend on each other's actions, which means that they cannot act alone, their actions must have a connection to each other.

Note : Sense/Think/Act is the performance of what the Romi will do. Sense means that it takes readings from the environment using sensors, think means that the microcontroller is doing the calculation/algorithm in the code, and act means the actions performed by the romi, usually by setting motor efforts.

Functionality	Sense/Think/Act	How we do it
Move back and forth: Command Motors	Act	Set the destination point for the Romi, then use Kinematics to calculate the effort. Give the effort to the motors. Romi will then drive to the destination point.
Standoff Control motors speed depends on the distance from the other romi	Sense, Think, and Act	An ultrasonic sensor measures the distance by measuring the time it takes for width length waves to return from the other romi. The standoff will take the distance readings to evaluate and give the calculated speed to the motors.

ii) Constraints and Challenges

For circling, our team decided to let Mercutibot move back and forth, then Tybot will follow his actions using an ultrasonic sensor. The main goal is to make sure Tybot will follow Mercutibot's movement, while they are performing the circling scene. We set the target distance between the two Romi's as 30 cm. Tybot needs to keep the distance from Mercutibot at 30 cm. The circling will not stop until Mercutibot is done with moving back and forth. Mercutibot will then update his state so Tybot can understand that the circling is over, and stop his circling. The state change gives an MQTT signal to Tybot to react this way.

iii) Options considered

Sensor	Pros	Cons
Ultrasonic	Easy to use, works well with Standoff, cheap.	Lots of noise, need a filter. Does not know what it's looking at.
OpenMV Camera	Fast response time, reliable.	More surrounding noise. We only have 1 camera. Expensive solution, for a simple task.
Sharp IR	Good stability over time, faster response time.	Cannot detect well within 6cm. Does not know what it's looking at. Not as reliable as the Ultrasonic.

With the options considered above, our first choice is still the ultrasonic sensor, as it works most reliably with our standoff code. Also it is easy to use, and cheap, so we do not have to waste more time on experimenting with other options with the standoff.

iv) Our solution

Our solution for the circling scene is to let Mercutibot drive back and forth two times using kinematics, and then Tybot will use the Ultrasonic sensor to follow along Mercutibot's movements using standoff. The target distance between the two Romis during the standoff is 30cm. When Mercutibot is done with his movements, he will change his state. Tybot will acknowledge the new state for Mercutibot, and change to another state to perform the next action. This is done using MQTT.

❖ *b) The stabbing act*

This is the scene when Tybot rushes and stabs Mercutibot. Romi will be updated with Mercutibot's death and prepare for his next action.

i) Functionality

Tybot stops his circling and begins to stab Mercutibot. After the stabbing is done, Mercutibot dies, and Tybot will prepare for his next stab scene.

Functionality	Sense/Think/Act	How we do it
Stab: Run into the other romi and stab them.	Act	Using forward kinematics set the motors' efforts to high values, and stop if there is collision detection.
Stab: Collision detection.	Sense	Using the IMU to detect a collision, based on the changes from the accelerometer.

Done stabbing: Stop motors and turn on the IR emitter.	Think	Stop the motors and set the IR emitter Tone, and to LOW when the collision detection is confirmed.
Die signal: Turns on LED.	Act	Red LED is turned on when the actor is stabbed.

ii) Constraints and Challenges

For this stabbing scene, Tybot runs into Mercutibot and Tybot must detect the "stab" to ensure he has stabbed Mercutibot. After the stab is done, Mercutibot and Romio both receive a signal from Tybot to know the stab is complete. The challenge here is the communication between the Romis, as there are three romis in the scene. The goal for the stabbing scene is that we have to make sure the collision detection works perfectly at the right time, when Tybot hits Mercutibot. After being stabbed, Mercutibot will turn on his red LED, as he "bleeds" all over the stage.

iii) Options considered

Options	**Pros**	**Cons**
IR Communication	Simple circuitry, low cost.	Can be affected by the remote signal in the room, alignment sensitive. Signal can be blocked by robot/environment.
MQTT (Wifi Communication)	No need for extra components or circuits. Works regardless of Romi alignment.	Time-consuming. Can break if there are multiple Romi's/other devices in the room.

After considering these options, our team ended up with the Wifi communication as it is reliable and we have already implemented it successfully. The IR communication can be blocked depending on what the yaw of the robot is.

iv) Our solution

When the circling is done, Tybot will run towards Mercutibot using kinematics, then hits him to create a "collision" for the "stab" scene. Afterwards Tybot uses the IMU to detect the bump by a change in the accelerometer. Tybot after detecting the bump will stop his motors, and send Mercutibot and Romio a signal through the MQTT server. To ensure none of the actors are damaged during the scene, we also attached a protection kit (made with foam and cardboard) for all Romi's.

❖ *c) The find Tybot and runaway act*

This is the scene where Romio will suddenly rush and stab Tybot, leaving another dramatic death on the stage. Romio will run away afterwards.

i) Functionality

Tybot turns on his signal when he has completed his stabbing scene. Romio will find Tybot and run towards Tybot to stab him, leaving Tybot "bleeding" on the stage with the red LED. Romio will then run away.

Functionality	Sense/Think/Act	How it's meet
Find the signal from IR emitter on the other Romi	Sense/ Act	Spin around to find the IR signal and stop then run into the place that transmits the signal
Stab Detect collision	Sense	Using the IMU to detect a collision, based on the changes within the y-coordinate from the accelerometer.
Run away Command motors	Act	Set the motors' efforts to turn around and run away by kinematics.
Die signal Turns on LED	Act	Red LED is turned on when the actor is stabbed.

ii) Options considered

Sensor	Pros	Cons
IR receiver	No need for wiring, easy to use.	Cannot know where exactly the ping is coming from, or how far it is coming from. It can receive a signal in a rather wide range.
IR camera	Based on the received ADC values, can determine how far the sensor is from the signal source. Accurate.	Can only detect the signal when the camera is placed in the range that it can find the IR signal.

We chose the IR camera for Romio to detect as it is easier to detect the emitter signal using the camera. As Romio will be somewhat far away, we want the received signal to accurately come from the direction we want to go to.

iii) Constraints and Challenges

The main goal for this scene is for Romio to successfully find Tybot, because the failure of this finding will lead to other failure for next actions. For the stabbing scene, we can again use the main function of the previous scene, but we have to make sure the stab is complete as Romio is stabbing Tybot from another axis direction. Another important target is to ensure Romio knows that Tybot has been stabbed. Through the IMU collision detection, and the MQTT, we can send the signal for him to run away from the dead Tybot.

iv) Our solution

Our team chose to use the IR camera for Romio to find Tybot after he stabbed Mercutibot. We modulated an IR Emitter to 38kHz and turned it on until Romio found Tybot. Romio has an IR camera and uses it to determine where Tybot is, by detecting the signal from the IR emitter. When Romio detects the IR signal, he will immediately rush into Tybot to stab him. Tybot has the IMU, so he can once again detect the collision, as Romio attacks him. Tybot, before turning on the red LED, will send Romio a signal through the MQTT server, then Romio will turn around and run off using kinematics.

3) <u>The Balcony scene</u>

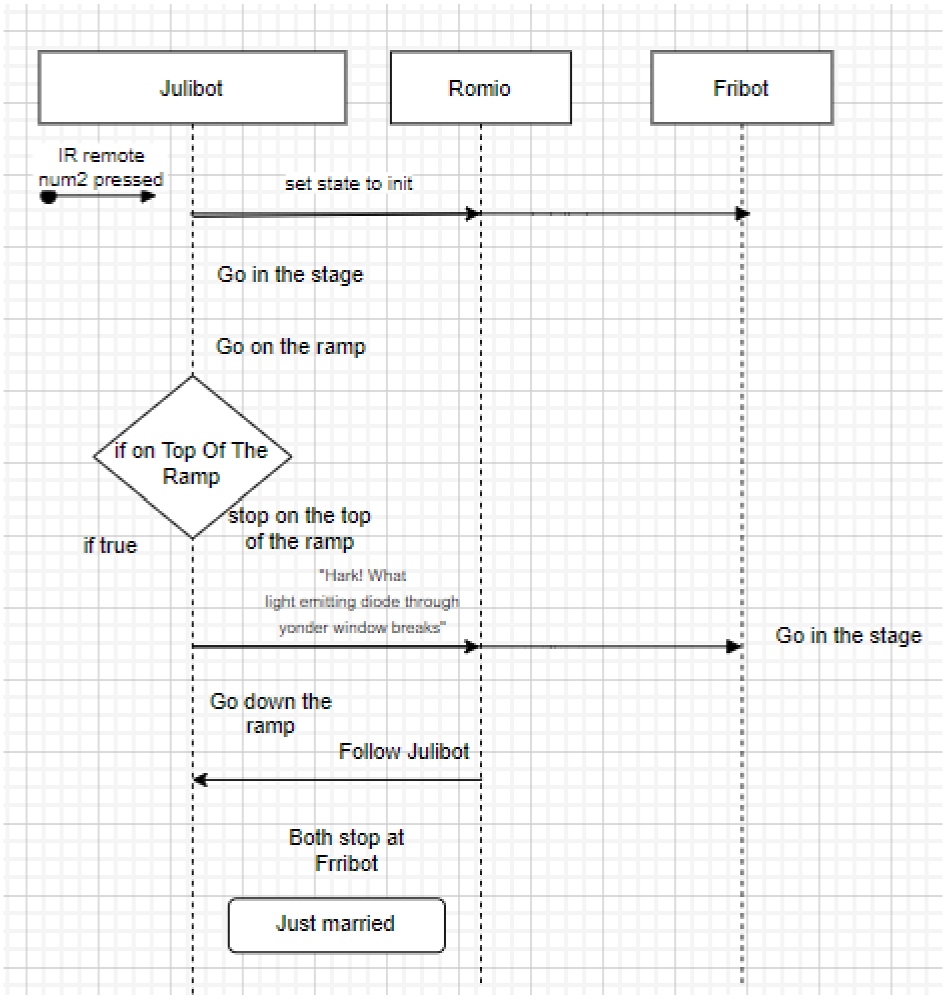


Diagram 2: Balcony scene motion diagram

The scene where Julibot climbs on the balcony, and leaves Romio a message to ask him to follow her. They will stop in front of Fribot to get married. And everyone will be happy.

- *a) The Julibot climbing on ramp act*

  The scene where Julibot enters the stage and starts climbing on the ramp. She will stop at the top of the ramp to send Romio a message.

  i) Functionality

Julibot will enter the stage and drive directly on the ramp, then she will stop at the top of the ramp to send Romio a message for the next scene.

Functionality	Sense/Think/Act	How it's meet
Goes on the ramp	Sense and Act	Goes on the ramp using wall following, by sensing with the Ultrasonic sensor.
Stop at the top of the ramp	Sense and Act	Detects the climbing by calculating the pitch angle using the IMU, and sets the motor speeds to 0 when at the top of the ramp..
Signaling to Romio to come to the stage.	Think	Send Romio a signal after she stops at the top of the ramp

ii) Constraints and challenges

It is important that Julibot has stopped at the top of the ramp, or else she will fall down. In order to make Julibot stop in the right place, it is also important for Julibot to climb up the ramp reliably, without missing the ramp. Julibot must also be able to send Romio a message when she is at the top of the ramp.

iii) Options considered

Options	**Pros**	**Cons**
Wall-following	More consistent for Julibot to climb up the ramp.	Have to know how far the ramp is from the wall. Need to establish PID control for following the wall consistently.
Forward kinematics	Easy to implement, no need for sensors.	If Julibot goes to the ramp at an angle, she will fall. Needs to perfectly set up the Romi towards the ramp.

We did try both methods, and ended up with the wall-following as it is more reliable. If the ramp is moved, we only need to change the target distance. With forward kinematics, we always need to align the Romi well, and hope we didn’t put it to a bad angle.

iv) Our Solution

Julibot will first go to the stage and do wall following with an Ultrasonic attached on her side. We will measure within 25cm of the wall. Then when she climbs up the ramp, Julibot will use her IMU to determine if she is climbing, using a hysteresis band to switch the boolean isClimbing to true. We calculate the pitch angle from the IMU accelerometer and gyrometer readings, using a complementary filter to determine if Julibot is climbing. Then we use the hysteresis band to switch the isClimbing boolean, and when we know that Julibot is on the top of the ramp. When that happens, Julibots motors stop. After she stops at the top, she will send Romio a message via the MQTT server.

- ❖ *b) Romio follows Julibot act*

  This is the scene where Romio hears Julibot's message and follows her while she is going down the ramp, then the two of them stop at Fribot to get married.

i) Functionality

Functionality	**Sense/Think/Act**	**How we do it**
Goes down the ramp using wall following	Sense and Act	Controlling the motor’s speed based on the distance readings from the Romi to the wall
Drive along/follow another Romi	Sense and Act	Adjusting the speed based on the OpenMV camera readings of the x-axle, and the size of the apriltag.
Stop at Fribot	Act and Think	We make Fribot bump into Julibot, and set off the IMU collision. Then, Julibot will tell everyone that she and Romio are married. Everyone stops.

ii) Constraints and challenges

The challenge is to make sure Romio has follows Julibot for the whole scene, as she is going down the ramp and beyond. If Julibot fails to go down the ramp, it is hard for Fribot to hit her and create the happy ending. It is also important that Julibot messages everyone once she and Romio are married. This is done using the MQTT.

iii) Options considered

Option	Pro	Cons
Wall-following	We have working, reliable wall following. Very little wiring needed. Easy to implement.	Hard to align Romio and Julibot to go side-by-side while Julibot is going down the ramp. Ultrasonic does not know what it's looking at, could try to follow something that is not Julbiot. Cannot center on Julibot well.
AprilTag + OpenMV camera	Knows who Julibot is with the apriltag, and will follow her and not something else. Can center on julibot with ease using the x-coordinates the camera gets from the apriltag.	Hard for Romio to be side-by-side to Julibot continuously. Expensive camera.

In the end, we use the AprilTag and OpenMV camera for Romio to follow Julibot while she is driving down the ramp. The camera can detect the x-coordinates from the apriltag, and center on Julibot. It can also detect how far Romio is away from Julibot and we can adjust how fast Romio can keep up with Julibot based on the tag size detected.

iv) Our Solution

Julibot will go down the ramp with an AprilTag using the wall following, and Romio will follow Julibot using the OpenMV camera. Based on the size of the tag on the camera, we can adjust how much Romio needs to turn to keep a constant distance from Julibot. And based on the x-axis coordinates, we can adjust the speed for Romio to keep up with Julibot. After going down the ramp, Julibot will stop as that is where Fribot enters the stage. Romio will also stop there. Then they get married.

4) The Curtain call scene

This is the final scene, where every Romi will come out of the stage and greet the audience as they drive out. Then they will return to the back stage.

- Our solution:
  We set different destinations for the Romi's based on which Romi it is. As the IR remote is pressed, one romi will communicate with the others through MQTT, and all of them will drive out together. The romis will spin towards the audience and blink the LED to represent the greeting to the audience. Then drive back to the backstage, putting the destination to (0, 0). They will stop, and turn off the LED's.

III. PERFORMANCE

Since our communication method heavily relied on MQTT, performing the scene in the lab sometimes interrupts our MQTT communication. The more devices there are, and the more Romi's use the MQTT, the less stable it becomes. Beyond that though, it was very reliable during our performances. During the fight scene, we had to reset a fair amount of times due to the MQTT disconnecting. There were more Romi's and people in the lab during that scene, so the MQTT was more unstable during the performance. For the balcony scene, our MQTT got disrupted at the beginning for a bit, but when it did run, our overall performance was quite smooth.

During the fight scene, we faced a fair amount of issues with Tybot and Mercutibot managing to hit each other. A lot of these problems came from when Tybot started rushing to Mercutibot, he had more power on one of the wheels, sometimes missing Mercutibot. Tybot didn't go fully straight towards Mercutibot and hit him directly. The standoff was another thing that was sometimes failing a little, as we couldn't fully center at Mercutibot, and could miss him a little. After Tybot had stabbed Mercutibot, Romio consistently got the MQTT call, and found the IR emitter with the IR camera, and ran at Tybot. Tybot's IMU collision detection also successfully ran each time. When Romio needed to run away, he also did that consistently and well. The main problems were with the circling of Tybot and Mercutibot, as well as Tybot running to stab Mercutibot. The problems we had eventually had to be solved by adjusting the PID for the motors and how they ran forwards. Another solution could've been to have the IR camera or OpenMV camera on Tybot, and something to sense from Mercutibot to have Tybot accurately see Mercutibot. Beyond this though, the performance went well and consistently. Even our LED lights went off always at the right time, there was no trouble with showing the Tybot or Mercutibot dying.

The balcony scene went even more smoothly, beyond the initial restarts due to the MQTT disconnecting on Romio. Julibot successfully went up the ramp in most cases, the wall following didn't really have problems. The IMU readings though could've used a better hysteresis band or a better filter to filter out unnaturally large readings. Sometimes we stopped before we got to the top, it wasn't a big issue, but it happened a little bit. Then Romio came in with the OpenMV camera, the camera successfully detected Julibot each time. The turning speeds were a little strong though, and Romio would turn a bit more than we wanted when going back. We needed to tune the PID controls for this a little bit more, and it could've gotten better. Still, it followed Julibot successfully through the program, Julibot coming down the ramp was never a problem while Romio was following her. Fribot coming out and bumping Julibot to initialize the marriage also wasn't a problem.

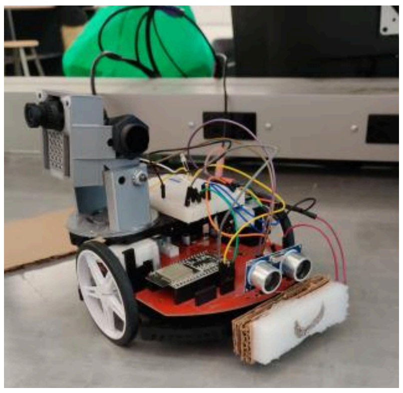
In conclusion, the demo went well and the balcony was a bit smoother than the fight scene, as we had to reset a few times in the beginning for the fight scene. The performance met our expectations, but if we noticed the problem earlier, we could have made the performance better. As our last thing, we also made the curtain call, which was very successful and didn't require restarts.

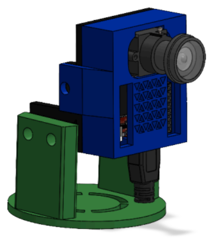
IV. ROMIS AND COMPONENTS

For each of the Romis, we attached and equipped the sensors and built parts so that it can perform its role at its best. Every Romi will have an ESP and DC motors, but the sensors on each romi are different depending on their role. They also have LED's as we use it before the scene begins to ensure the Romi's are connected to the MQTT server (the LED is on when it is connected). But during the scene, only Tybot and Mercutibot use it to communicate that they are dead.

ROMIO

Romio has an Ultrasonic sensor, an IR camera and an OpenMV camera. Romio uses the IR camera for the first scene where he detects Tybot and rushes into him, the OpenMV camera is used in the balcony scene to follow Julibot. The Ultrasonic is not eventually used for Romio, but is there as a backup for detecting Fribot and Tybot in the different scenes.





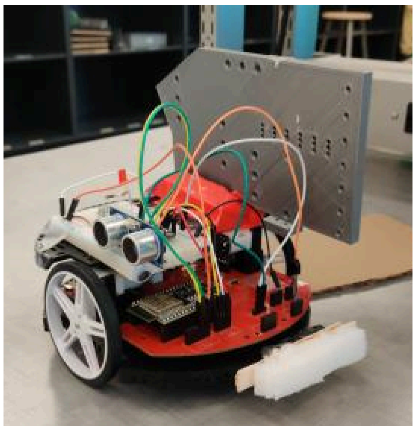
Picture 2 : Romio

To ensure that both cameras are placed in the right position and for consistency, we implemented a 3D CAD model for the camera, which is used in the demo to hold both of the cameras. We also give him a bump protection made of foam and cardboard, as he has to stab Tybot in the fight scene.

TYBOT/JULIBOT

Tybot has an Ultrasonic sensor (in front), an IMU and an IR emitter. He uses the Ultrasonic sensor (in front) to do the circling with Mercutibot, and the IMU to detect both Mercutibot's death and his own death in the fight scene. Tybot then uses the IR emitter to transmit a signal to Romio, so Romio can find him and perform the stabbing scene.
Julibot also has an Ultrasonic sensor (to the side) and an IMU. She uses the Ultrasonic sensor to do the wall-following, as she goes up and down the ramp. She also uses the IMU to detect the climbing to stop at the top of the ramp.

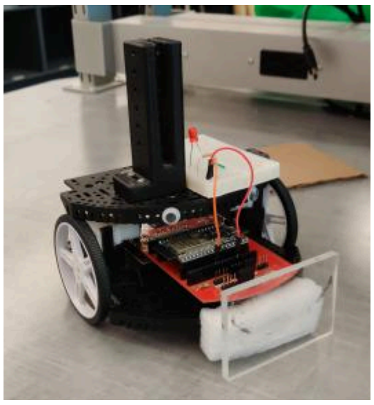
Since both Tybot and Julibot need an Ultrasonic sensor and an IMU, it is a good decision to let the actor share the two roles. For the ultrasonic, after the fight scene, we have to move it to the side for the balcony scene.



*Picture 3 : Tybot/Julibo*t

To make the placeholder for the Apriltag, we also have a simple 3D printing of a wall attached to Julibot, so she can lead Romio in her scene. We also give Tybot a bump protection kit made of foam and cardboard, so he can stab Mercutibot in his scene.

MERCUTIBOT/FRIBOT



Picture 4: Mercutibot/Fribot

Mercutibot/Fribot does not have any sensor components, as they do not do as much in the scenes. Mercutibot has an arcylic piece in front of him to make sure Tybot can “sense” him and do the circling. He also has an LED to show that he is bleeding.

V. APPENDICES

Team Member:	Percentage Contributed:
(Leona) Nhi Nguyen	33.3%
Tuomas Pyorre	33.3%
Theodore Winter	33.3%

Each persons contribution:
(Leona) Nhi Nguyen: Worked on the IMU collision detection, as well as other multiple aspects of the Romi's, like the cameras, the ultrasonic distance sensing, the complementary filter, and the pose of the robot.

Tuomas Pyorre: Set up the MQTT communication between romi's, as well working on other multiple aspects of the Romi's, like the wall following, wiring, OpenMV camera standoff, and robot pose. Also formatted and commented a lot of the code for readability, as well as finished the curtain call scenes.

Theodore Winter: Worked on the pose and the IMU collision detection, as well as other multiple aspects of the Romi's, like the PID for the cameras, the complementary filter, and setting up the switch case base for the final project.

Link to our GitHub repository : https://github.com/RBE200x-lab/RBE2002Code11

Here is the final release on GitHub:

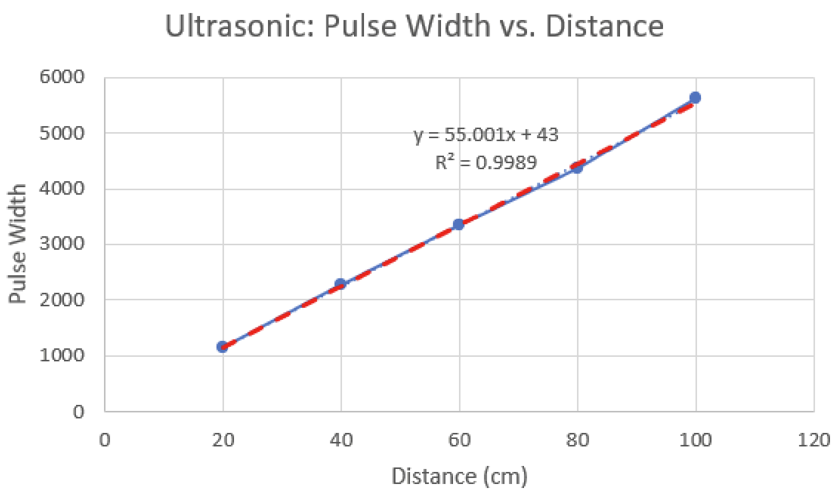
https://github.com/RBE200x-lab/RBE2002Code11/releases/tag/v.2

We can state that every member in the team contributed a great amount of work into the final project to make the stage scenes as great as possible. The code on Github is pushed but in reality, whenever we meet to do the problem together, we always live share our code in Visual Code Studio.

Apart from the code, here are some details about how we implemented the code using the sensors and each method mentioned in the solution section.

1) <u>Rangefinder (Ultrasonic HC-SR4)</u>

For using the Ultrasonic sensor (HC-SR4), we connected it to pins 16 and 17 on the Romi board and used them to send a pulse and receive it, then use it for calculating the distance.

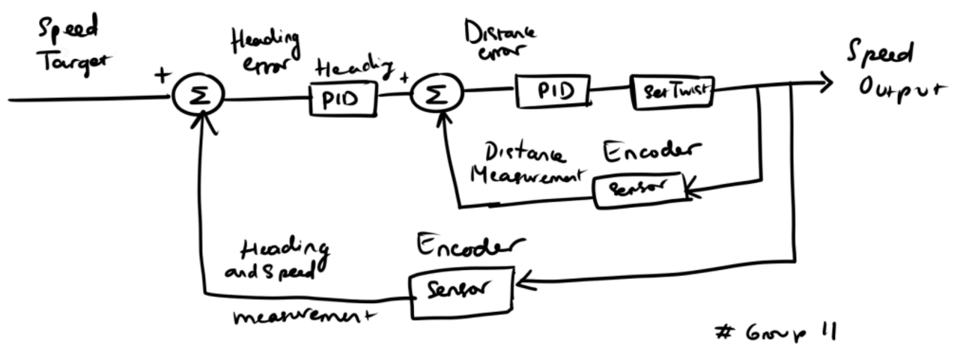


Graph 1 : Pulse width vs distance reading using Ultrasonic sensor

2) <u>ICC Kinematics</u>

For the forward kinematics, we use the ICC kinematics to determine the pose using motors' encoder count and speed, then set the destination point before telling the Romi to drive to that point. The idle point for every start is (0, 0).

[BLOCK DIAGRAM DRIVE TO POINT]



We choose the Instantaneous center method as it performs the arc smoother, and can perform a complete arc, without changing much of the theta, and because we have a differential drive robot the ICC is the best choice for us.

3) <u>Acceleration/Gyrometer complementary filter</u>

The equations and implementation can be found in the code.

When we set k = 1 in the filter, the output pitch angle completely depends on the gyroscope raw readings, it will drift, while when k = 0, the output pitch angle is completely dependent on the accelerometer raw readings, there will be a lot of noise.

Our final value of k is 0.875 and we determine this value by experimenting with the k values around 0.5 to 1 and it works best with 0.875. The k value leans to 1 more than 0, which means our filter is leaning to use the gyroscope values more than the accelerometer values. The gyroscope raw readings are more reliable (as they are more stable) and it has less noise than the accelerometer raw readings.

Any questions about the report can be emailed to *ttpyorre@wpi.edu*| *nnnguyen@wpi.edu* | *tdwinter@wpi.edu*