Final Report on Robô Jogo Bonito

Ahilan Balakrishnan, Hunter Greenlee, William Pope, Hemant Sethi
MCEN 4115/5115: Mechatronics and Robotics
22 July 2022

***Abstract*—Robô Jogo Bonito is an autonomous ground robot designed to compete in a penalty shootout competition.**

I. INTRODUCTION

This project involved the development of a robot that autonomously played offense and defense in a soccer penalty shootout scenario. In offense, the robot needed to kick a 7" diameter ball past an opponent goalie into the goal. For defense, the robot had to intercept the ball and keep it out of its goal.

II. DESIGN

A. Shooting Mechanism

The kicking mechanism was inspired by the ease of torque transfer that happens seamlessly on a bicycle between sprockets and bike chain. This would provide the main concept for rotating a shaft with a high amount of torque now we just needed potential energy from the spring which is discussed further in the report. In order to hit the ball a foot was created from steel tubing. The added weight of the steel foot provided some extra force for making contact with the ball. Connecting the foot and the rotating shaft together provided somewhat of a challenge but was met with the solution of screwing the foot to the leg and leg to the rotating shaft. This process was helped by the use of ¼-20 thread cuts which is further discussed in the machining section of the report. The short summary of this kicking mechanism is as follows, a rotating shaft set screwed to a sprocket given its force and energy from a spring, a connecting leg set screwed to the foot and shaft and finally pillow blocks to allow an unopposed moment and free rotation.

When selecting a spring, we looked to optimize over a tight set of constraints. The first was that the spring could only extend to about nine inches due to the size of the robot's chassis. The second was that the force applied to the spring needed to be low enough that a human could load the kicker, and that the force wouldn't cause the robot's structure to collapse. We chose a force of 20 lbs / 90 N for this.

The goal of optimizing the spring was to produce the fastest shot possible, which meant maximizing the energy transferred to the ball. The key equations we used in our search were:

- Kinetic energy of the ball: $K = \frac{1}{2} m v^2$
- Coefficient of restitution: [CoR eqn]
- Potential energy in the spring: $P = \frac{1}{2} k x^2$
- Force on a spring: $F = k x$

By simulating different initial velocities for the ball, we found that in order to arc a shot 12 inches high (over the goalkeeper's chassis), the ball needed [v_0], which corresponds to a kinetic energy of [K]. When the striker contacts the ball, some energy is lost in the collision, which is characterized by the ball's coefficient of restitution. This coefficient can be calculated by dropping the ball from a set height and measuring how high the first bounce returns, to determine energy lost in the bounce. Testing this on the tile in the lab, we calculated that the rubber ball used has a CoR of [CoR], which is fairly high. This meant that the kicking mechanism needed roughly [1/CoR * K] in order for the ball to receive the required energy. In turn, this meant that the spring needed to store at least that much energy when loaded, with some margin for energy lost in the mechanism.

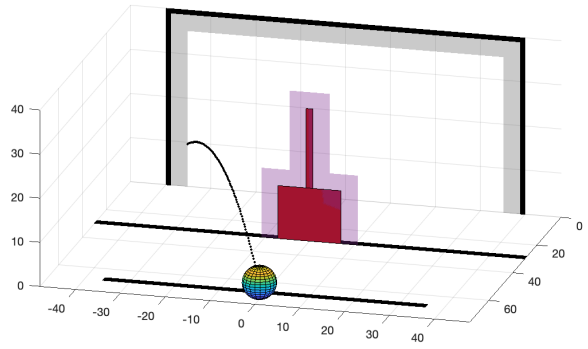


Fig. 1: Desired shot trajectory

By stepping backwards from a desired shot trajectory, we were able to set a rough target for the spring's energy when loaded.

B. Chassis

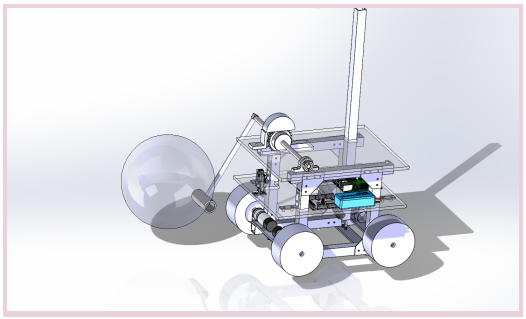


Fig. 2: SolidWorks CAD Model

The chassis was built from ¾ in aluminum square tubing and steel 90's to connect them. The reason aluminum was necessary is because for our first calculation made by Will suggested the spring would create a ~25lb force that an acrylic chassis would not be able to withstand. Because our CAD model was so in depth we were able to manufacture everything no matter how long it took with great precision. Some parts we were unable to obtain or create sturdily with aluminum as we would have liked, such as the motor mounts and the adapters from our wheels to the smaller shaft of the high RPM motors. So we instead 3D printed these parts and worked with them the best we could.

The laser cutting was the most seamless and easy to implement part of our chassis. Because we spent so long on the CAD we knew where all the wires, holes for arduino and RaspberryPi, Pixy mount, brackets, and various other through holes needed to go. All we really had to do was send the DXF file and plop down a board and let the machine do its magic.

A Final Note on the Chassis: Although the manufacturing process was at times challenging for the chassis I believe that it was crucial to our overall success because it provided a steady and unwavering base for our robot as well as legitimate rigidity for the forces from the springs.

C. Drive System

The ability to move freely about the floor was highly valued for both offense and defense. We considered several choices of wheels and motors, including:

- 2-wheel differential drive (Roomba-style)
- 4-wheel differential drive (Jackal-style)
- Car-like steering
- Parallel mecanum wheels
- Orthogonal omni-directional wheels

We ruled out differential drive and car-like steering because the vehicle can only move in the direction of the wheels at any given time, making maneuvering more cumbersome. Traditional orthogonal omni-directional wheels were ruled out as well, because they require the motors to point towards the center of the vehicle, taking up more internal space. With other options eliminated, we selected mecanum wheels.

The next consideration was the size of the wheels. A larger wheel covers more ground in one revolution, but requires more torque to get moving. The primary motivator in wheel selection was the speed of the vehicle, since we needed to quickly move towards a shot when defending the goal. This incentivized a larger wheel diameter. Our initial motors topped out at just 60 RPM, so we chose the largest wheels easily available/afforded on Amazon: 97 mm.

One drawback of mecanum wheels is that the vehicle's speed is limited to 70.71% of the wheel's

actual speed, due to the 45 degree diagonal rollers on the rim of each wheel. To remedy this, we switched to more powerful DC motors with a higher speed of 130 RPM. We later went even further with a gearing that allowed for 300 RPM. However we later saw that the higher speed gearing did not produce enough torque to move our robot's mass, so we reverted to the 130 RPM motors.

D. Sensors

Sensing and detection were important aspects of the robot's ability to move autonomously. Various sensors were used to provide ample data to the "brains" of the robot to calculate and execute decisions efficiently.

Initial ideas revolved around a sophisticated object detection camera system, two range sensors to triangulate object positions, reflective sensors to prevent the robot from going out-of-bounds, and a microphone sensor to detect the start whistle (2093 Hz). In the final design, some ideas were deemed to be on the ambitious side and the sensors were cut to just using a PixyCam (Pixy2) and a microphone, all connected to the Arduino Mega.

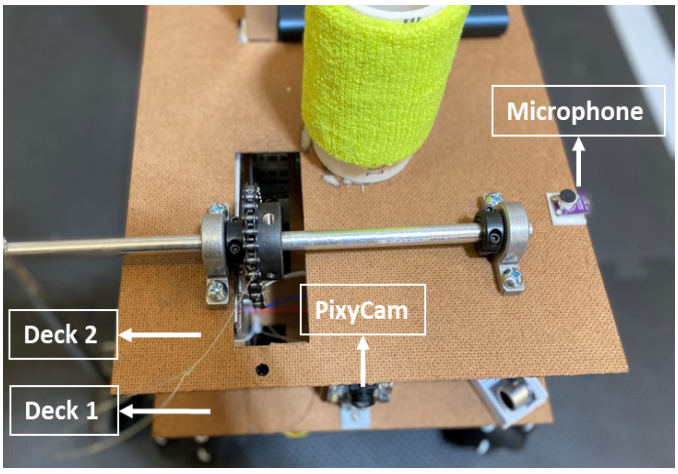


Fig. 3: Location of Sensors - PixyCam and Microphone on Deck 1 and Deck 2 respectively

The Pixy2 was attached to the front and center of the robot via a custom 3D printed mount. It was placed on the first deck at a height of 4.5" from the ground, providing it a good view of its environment and most importantly, the ball. On the software side, the setup was simplified via the PixyMon app. The camera was taught to look for signatures of the red ball, the opponent bands, and the two uniquely colored boards on either side of the goal posts. The sensitivity of the color recognition program was calibrated on the playing field while using the overhead lights. This helped avoid brightness and contrast differences. Additionally, the pixy2 was hardcoded to only detect one object (block) for each color signature. Both steps helped minimize false positives while providing x and y coordinates of the detected objects at 60 frames/second.

The microphone amplifier (Maxim MAX4466) was used to detect the 2093 Hz frequency as the start-whistle of each game. This served as a cue for the robot to switch modes (refer to section III) and execute a given set of operations. The location for the mic was important as the amplifying sensor was highly directional. After some testing in a closed space, it was attached on the right side of the top deck facing the ceiling. The idea was to catch the frequency bouncing off the ceiling such that it could be heard for both offense and defense modes.

This method largely worked during testing as we were in relatively closed spaces, however the robot did have some issues detecting the mic while on the playing field (slightly more open). During defense, when the mic was closer to the sound source, detection worked flawlessly. However, during offense mode, when the mic was away from the source, the robot had some issues. To avoid this issue in the future, multiple sensors could be used or, location can be changed to right above the PixyCam but on the topmost deck.

E. PCB, Wiring and Electronic System

To remove the more junky wire connection we plan to make a PCB board with the help of Eagle CAD .Initially, we plan to design the 2 layer PCB but we end up doing 1 layer PCBboard to mainly reduce the cost and reduce the manufacturing time .

We make the PCB mainly for integrating two motor drivers and for relay to control the solenoid. Fig 4 shows the schematic of our PCB.

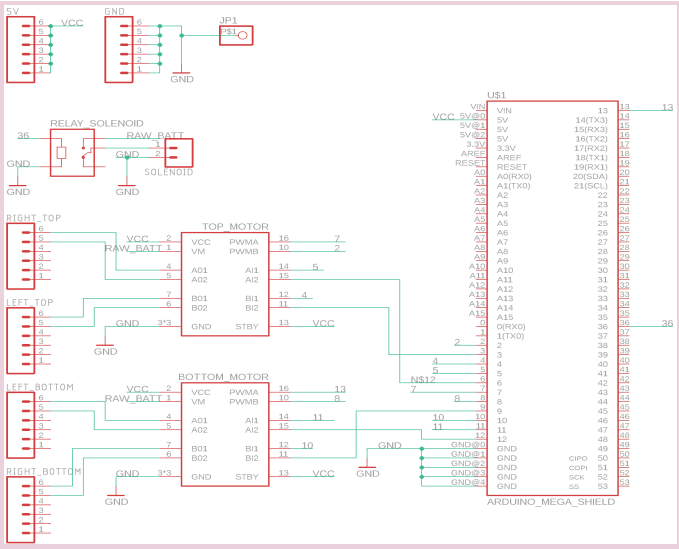


Fig. 4: Schematic of PCB

We have considered the following points while designing the PCB,

- The motors driver and Relay have to work on 12V DC supply so we make wire routing thickness increase to 40 mill to handle that.
- Motor driver having PWM connection with arduino so to avoid the signal interference we shorten the wire routing and cover this with ground plane

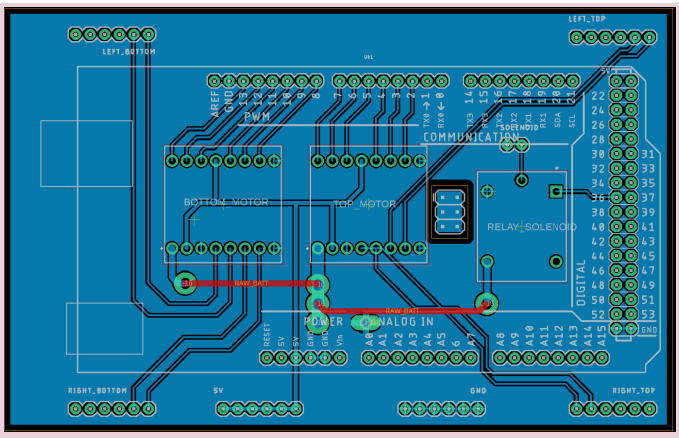


Fig. 5: Final board design

F. Computing

In order to control our vehicle, we used both a Raspberry Pi 4 single-board computer and an Arduino Mega microcontroller. The two boards worked together to handle tasks better suited to each.

The Arduino is used at a lower level for controlling all of the robot's sensors and actuators. With four sensors and five actuators on board, the Pi simply didn't have the GPIO ports to connect to everything. In general, a microcontroller is better suited for this low-level device management, so it was selected to be the main hub for all peripherals.

It is likely that the same Arduino could have been used for high level computing as well, running all of the robot's operations for offense and defense. However, we selected a Raspberry Pi for the job, largely for the convenience of Python. Early software ideas included more complex tasks like Kalman filtering and game theory, which would have been more suitable for Python. In this arrangement, the Pi acted as the central decision-maker for the robot, while the Arduino ran a simple script to interact with the sensors/actuators when commanded by the Pi.

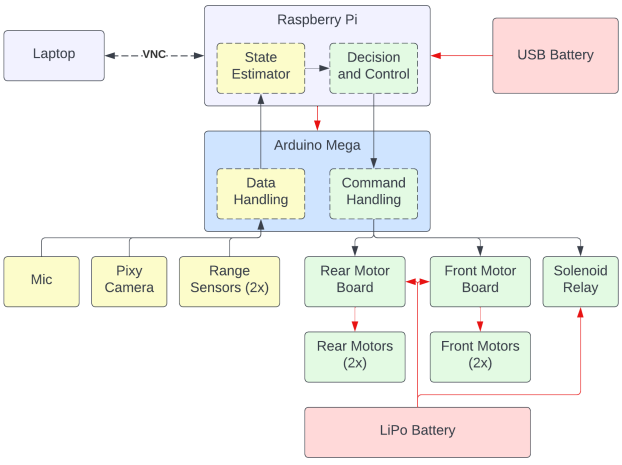


Fig. 6: System architecture

G. Communication

The Pi and Arduino acted as primary and secondary, and passed data back and forth using the UART protocol. This is implemented with the pySerial and Serial libraries. At each time step during operation, the Pi requests sensor data from the Mega, which queries the sensors and returns the data to the Pi. From this observation, the Pi decides on what action to take. The Pi then sends an actuator command to the Mega, which the microcontroller acknowledges and executes.

The UART protocol does not designate a primary and secondary, so the Arduino script is set up such that it does not perform any tasks or send any data until it has received a request from the Pi. There are functions on both the Pi and the Mega that assemble/disassemble variables into individual bytes for transmission. The largest packet is the sensor observation data, which takes 25 bytes to represent measurements from two range sensors, four Pixy objects, and the microphone. At a baud rate of 115200 bits/sec, this packet is transmitted in just 1.7ms.

H. Control

The vehicle is controlled using an observe-decide-act loop, where it constantly observes the environment through sensors, decides a new action based on the observation and current objective, then executes that action using the actuators. To create context for observations and actions, unique discrete modes are created for each phase of the mission. These are described fully in Section III. The transitions between each discrete mode are based on specific observations, for example, Mode 0 transitions to Mode 1 when the whistle is heard.

On offense, the robot needs to position itself near the ball to take a shot. On defense, the robot needs to place itself between the ball and the goal in order to block the shot. Both of these cases make use of closed-loop feedback control to accurately reach the desired position. For best results, the time between the observation, decision, and action should be as small as possible. We chose a nominal action rate of 4 Hz, which allows enough time to query each of the sensors and pass data back and forth between the boards.

III. CONCEPT OF OPERATIONS

Our strategies for offense and defense evolved over time, generally becoming less complex as the limitations of our hardware became apparent. This section will describe both the ideal strategy that we designed for and the more realistic strategy actually implemented.

A. Offense

Because the defending robot's area is largely below 12 inches, we planned a kicking system that could launch the ball into the top corner of the goal–a nearly guaranteed goal. This proved to be unrealistic due to the limited energy provided by the spring.

Our initial strategy was to track where the goalie was, then line up a shot towards the opposite side of the net. However this approach required the ability to track the goalie, move the robot to a new shot angle, and move the robot into position to take the shot. We didn't have time to fully realize this method, so we instead went for a simpler approach where a team member placed the robot at a fixed position/angle, so that it only needed to take the shot.

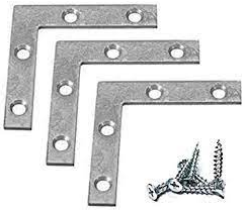
B. Defense

For defense, we used the body of the robot to block incoming shots. The vehicle needed to track the ball and move towards it, accomplished with the Pixy camera and omni-directional wheels. The lateral speed of the robot was proportionally controlled according to the coordinate of the ball in the camera's frame. This meant that the robot moved at high speed when chasing the ball, but low speed once in position for a block.

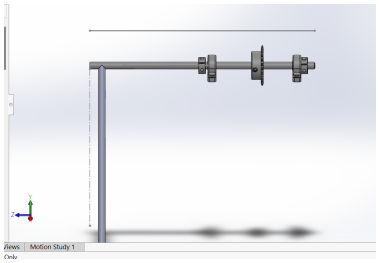
IV. BUILD

A. Machining

The machining of our robot was to enable the use of M3 screws to rigidly attach our chassis pieces. Although we machined our own corner brackets this image provides a valid representation of what was done on our robot. The holes that we needed to drill included four holes on each vertical bar to match the machine screw holes on our corner bracket and then an additional two holes to mount the laser cut board to the robot. On the bottom horizontal bars an additional 4 more holes were drilled for the corner brackets, then on the bottom bars (¾ in square aluminum) 2 holes were drilled for attachment of the motor mounts and finally an additional 2 for the aluminum cross bar on which the servo motor and loop for the spring sits. The final two bars on the top of the robot required holes to attach the pillow blocks and top laser cut board. All of these holes were drilled on the drill press and marked out with calipers.



The kicking mechanism also had a considerable amount of machining in order to handle the torque created from the springs. In order to attach the rotating shaft to the leg as seen on the left, threads were placed through the rotating shaft and ½ inch into the leg. This was done on the lathe and then a hand tap. The same concept was used in order to attach the sprocket to the rotating shaft (threaded through hole in the rotating shaft for the ¼-20 screws already in the sprocket). Finally to attach the foot of the robot to the leg threads were created in the bottom of the leg. The foot on top had a hole 7/16 in thick so the ⅜ shaft could slide through and a 5/16 in hole on the bottom that a ¼ -20 screw could slide through (but not the head in order to be able to tighten the screw to the leg). The holes on the foot were all done on the drill press.



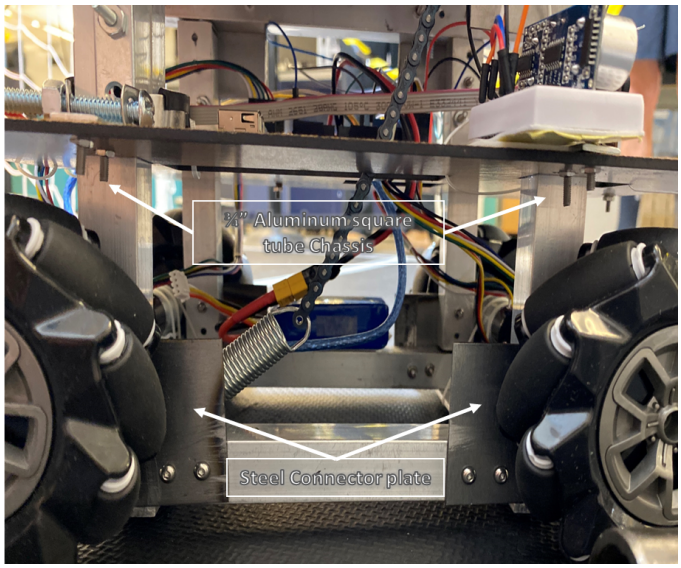


Fig. 7: Side View of Robot

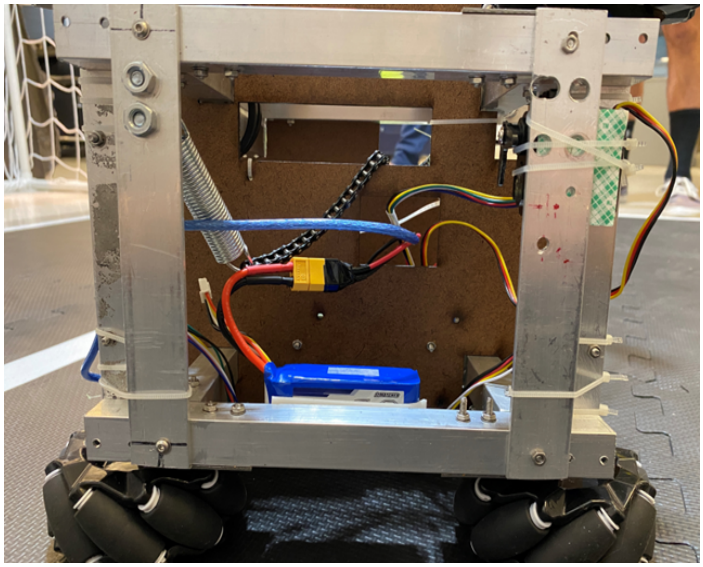


Fig. 8: Bottom view of Robot

B. 3D Printing/Laser Cutting

For 3D printing and laser cutting, designs were first established via CAD on SolidWorks. PLA was used to 3D print the motor mounts, adapters to fit the mecanum wheels to the motors as well as the PixyCam mount. The infill % was changed from 50% to 75% to make the mounts stronger while the layer height was kept constant at 0.3mm. During the

testing of the robot, several mounts failed, likely due to a small fillet radius. Due to the shortened equipment hours, more mounts could not be printed on time and zip ties were used to hold the parts together as shown in Fig. 9.

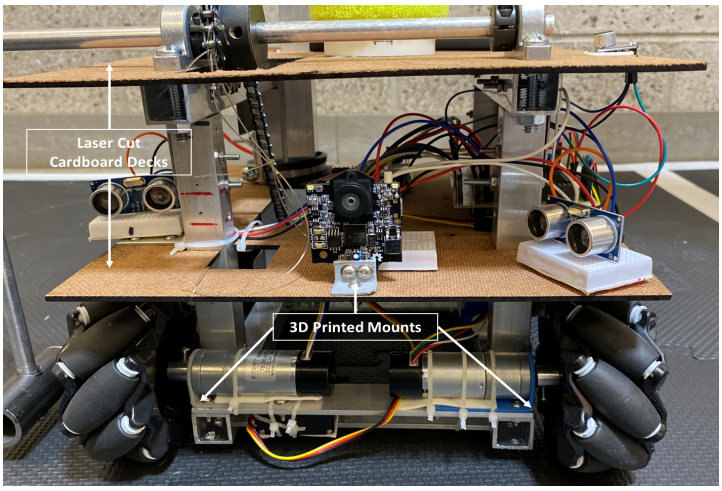


Fig. 9: Front view of Robot showing 3D printed parts and laser cut decks

The decks of the robot were made of ¼" thick high density particle boards and were cut using laser cutters. The "print" settings used were 75% power and 20% speed in the vector cut mode. It resulted in two precisely cut decks that provided good strength and structure to the robot in addition to real estate for the various devices and equipment.

C. PCB Fabrication

Eagle CAD design is used to make PCB design and from that PCB was fabricated. We used a long female header pin to make easy connections with the Arduino and the male pin for connecting the motors as shown in Fig. 10. As a precautionary measure, we soldered the wire for battery connection separately. While mounting the PCB on the Arduino, metal parts were covered with an insulation tape to avoid static current on the PCB Board.

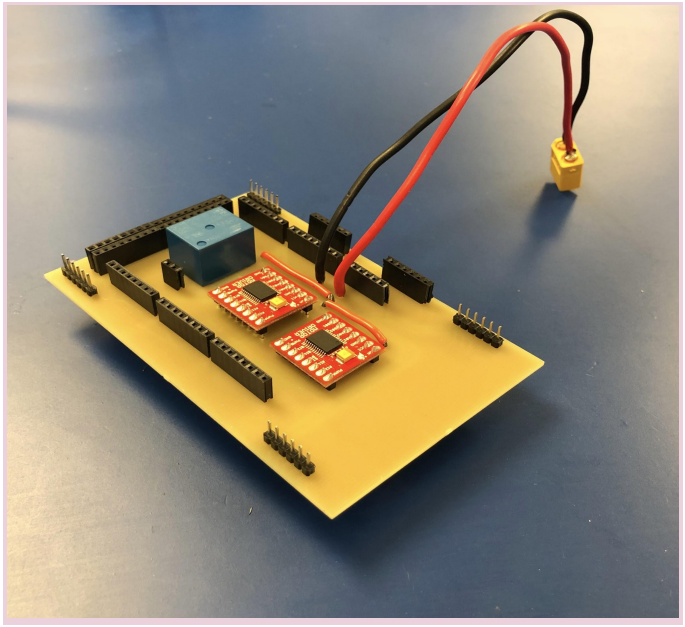


Fig. 10: Final PCB after assembly

D. Sensor and Actuator Calibration

First, the observe-decide-act loop time was brought down from ~500um to less than 200um by making parts of the code more efficient during defense. The primary data needed for making decisions were the x,y coordinates of the red ball. Requests for all other sensor data were put in "sleep-mode" and skipped over to help with the loop's efficiency.

Second, a calibration algorithm was developed to avoid the robot from moving erratically as new data/command passed through to the motors after processing in a feedback controlled loop. Essentially, if the ball was seen within a window of ± 25 pixels from the center, the motors were throttled to 25% of the speed in order to make definite contact with the ball and not pass it. However, if the ball was outside that range, the actuators were provided with full power to attempt and make contact with the ball.

This calibration algorithm was tested with several variations of the pixel window and finally we settled with ± 25 pixels. This worked great for us and our robot as our team had the best defensive record and allowed only 4 goals in 5 games.

V. NOTABLE ISSUES

A. *Machining Complexity*

An issue that arose in this project was the time sink that machining ended up being. The time taken in order to mark out each cut with calipers, scratch alls, and markers then clamping the material down for the drill press, drilling pilot marks for the drill bit and finally drilling the holes themselves took a very long time which could have been used for the code or the release mechanism. I think that our design could have been simplified to some extent but was also crucial to the rigidity of our robot which was necessary to resist the high forces created from the spring. The vertical bars also made it easy to attach the laser cut boards on the middle and top layers.

B. *Inter-Device Communication*

In order to pass data between the Pi and the Mega, they needed to be connected with some wired communication scheme. We initially chose to use UART, since we could just use the USB cable running between the two devices. However this proved difficult to implement, because the Arduino was unable to print to the Serial Monitor when communicating with the Pi. This greatly reduced visibility into the Mega, making debugging a challenge.

We then switched to I2C, since it allowed for easier debugging and had convenient request/receive interrupts defined in the libraries. However, I2C seemed to be much more susceptible to interference and voltage issues, as we frequently received board-level errors when trying to use the protocol in our integrated robot.

After trying different fixes, we eventually switched back to UART in the last week of the semester. Once the basic write/read functions were working properly, code on the Mega could be debugged by just sending the desired data back to the Pi to view.

C. *Solenoid to Servo switch*

For the spring/chain release mechanism, a push-pull type solenoid was our first choice. It worked well with lower loads however the 5N pull was not able to overcome the metal-to-metal friction with the U-bolt that held most of the spring's load. Lubricants like Molykote and 3-IN-ONE oil to decrease the friction but ultimately this design had to be scrapped and replaced.

New mechanism featured a servo motor with a V-shaped attachment. The attachment was screwed in with loctite to prevent it from loosening while in operation (see Fig. 11). The loading position had "the V" at 180° where the sharp notch held the load of the spring via a 40 lb fishing line. The V shape rendered the moment of the load to zero and held the mechanism in place even at high loads. To activate the mechanism and release the spring, the servo motor was set to 0° which released the fishing line and consequently the spring/chain kicking mechanism.

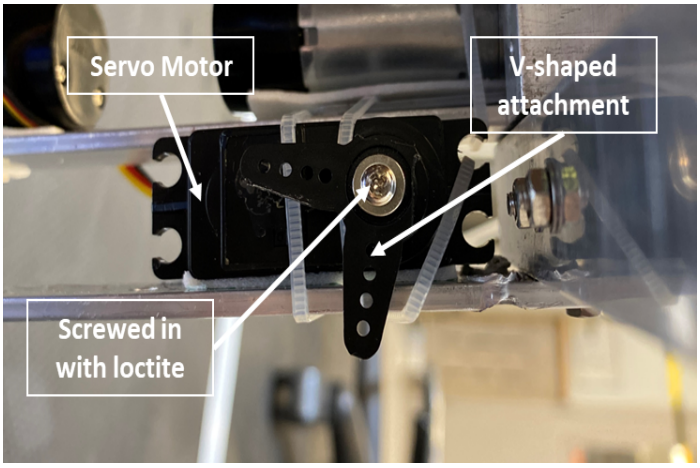


Fig. 11: Servo Motor with V-shaped attachment

VI. PERFORMANCE

A. Initial Testing

Initial testing involved a lot of trial and error in various aspects to get the best performance out of the robot in the given time frame.

On the defensive side, as mentioned earlier, the defensive ball tracking algorithm was calibrated and it worked well once the pixel window was decided and applied.

On the offensive side, the spring extension was chosen (per the calculations) such that it also did not exceed the max torque of the servo motor (3.3 kg.cm).

Next, the complete loading of this kicking mechanism was practiced and tested several times in order to conform to the 1 minute time limit between kicks.

Finally, the placement of the robot for the best contact to lift the ball off the ground was practiced along with the optimized kicking angle to hit the side nets.

B. Competition Results

Robo Jogo Bonito was off to a rocky start in the seeding tournament because the microphone stopped registering the start whistle in offense mode. However, it was strange to notice the mic working flawlessly during goalie (defense) mode. So, with a great display of lockdown defense, the team was able to pull off three draws and conceded only 1 goal in the process putting RJB in the 4th seed.

At the same time, the team worked to fix the microphone issue between games. Since the mic trigger failure only occurred during offense mode i.e. when the mic was facing away from the whistle, it was deemed that the open space of the play area, as well as the location of the mic, were both partly responsible for the issue. As a quick solution, a small speaker was placed beside the microphone to aid in triggering during offense. This plan was tested, put into action, and it provided a consistent trigger boost for the offensive mode.

During the playoffs, the offensive and defensive sides worked great and RJB was able to beat the 1st seed team **{2-1}** to book its spot in the finals of the robo soccer tournament.

The final initially ended in a **{2-2}** draw and was extended to a golden goal. The opponent shot hit the post and went out while the ***RJB bot scored to win the tournament***.

Overall, the RJB robot's record in the tournament was:

{2} Wins,
{3} Draws, and
{0} Losses.

It conceded 4 goals and scored a total of 6 in 5 games.

VII. CONCLUSION

The robot was an overall success. It did well in offense with the novel polo-style kicking mechanism. It was able to lift the ball off the ground and get some bounce when shot towards the goal. On defense, the ball tracking algorithm worked wonders to prevent shots from entering our goal. The faster 130 RPM wheels provided ample torque and speed to intercept most shots.

There are several aspects of this project that can be worked on and improved. On the software side of offense, RJB initially planned to autonomously detect the ball using range sensors, close in on it, calculate the best kicking angle and perform the kick. Similarly, on the hardware side, a spring with a higher spring constant (k) could be used for increased shot power.

VIII. REFERENCES

- Mark Rober's Field Goal Robot
- Project GitHub Page
- Data Sheets of parts mentioned in Bill of materials.

IX. APPENDICES

A. PinOut Diagrams

Self Device	Placement	Self Port	Partner Port	Partner Device
Raspberry Pi	Middle Deck	USB-B	USB-A	Arduino Mega
Arduino Mega	Middle Deck	USB-A	USB-B	Raspberry Pi
Motor Driver Bottom	On PCB	Motor Voltage (VM)	VIN	LiPo
		Logic Voltage (VCC)	5V	Arduino
		GND	GND	LiPo
		Channel A Output 1 (AO1)	12V Input	Motor 1
		Channel A Output 2 (AO2)	GND	Motor 1
		Channel B Output 1 (BO1)	12V Input	Motor 2
		Channel B Output 2 (BO2)	GND	Motor 2
		GND		
		GND		
		Channel B PWM Input (PWMB)	13	Arduino Mega
		Channel B Input 2 (BIN2)	12	Arduino Mega
		Channel B Input 1 (BIN1)	11	Arduino Mega
		Standby (STBY)	5V	Arduino Mega
		Channel A Input 1 (AIN1)	10	Arduino Mega
		Channel A Input 2 (AIN2)	9	Arduino Mega
		Channel A PWM Input (PWMA)	8	Arduino Mega
Motor Driver Top	On PCB	Motor Voltage (VM)	VIN	Arduino Mega
		Logic Voltage (VCC)	5V	Arduino Mega
		GND	GND	Arduino Mega
		Channel A Output 1 (AO1)	12V Input	Motor 3
		Channel A Output 2 (AO2)	GND	Motor 3
		Channel B Output 1 (BO1)	12V Input	Motor 4
		Channel B Output 2 (BO2)	GND	Motor 4
		GND		
		GND		
		Channel B PWM Input (PWMB)	7	Arduino Mega
		Channel B Input 2 (BIN2)	6	Arduino Mega
		Channel B Input 1 (BIN1)	5	Arduino Mega

		Standby (STBY)	5V	Arduino Mega
		Channel A Input 1 (AIN1)	4	Arduino Mega
		Channel A Input 2 (AIN2)	3	Arduino Mega
		Channel A PWM Input (PWMA)	2	Arduino Mega
Pixy Camera	Middle Deck, wires route to ICSP pins	ICSP SPI	ICSP SPI	Arduino Mega
Range Sensor 1	Middle Deck, wires route to I/O pins	VCC	5V	Arduino Mega
		Trig	31	Arduino Mega
		Echo	32	Arduino Mega
		GND	GND	Arduino Mega
Range Sensor 2	Middle Deck, wires route to I/O pins	VCC	5V	Arduino Mega
		Trig	31	Arduino Mega
		Echo	33	Arduino Mega
		GND	GND	Arduino Mega
Microphone	Top Deck, wires route to I/O pins	VCC	5V	Arduino Mega
		Analog Read	A0	Arduino Mega
		GND	GND	Arduino Mega
Servo Motor	Lower Deck, wires route to I/O pins	VCC	5V	Arduino Mega
		Analog Read	45	Arduino Mega
		GND	GND	Arduino Mega
Wheel Motor 1	Lower Deck, wires route to PCB	Motor 12V Input	12V Input	PCB
		Motor GND	GND	PCB
Wheel Motor 2	Lower Deck, wires route to PCB	Motor 12V Input	12V Input	PCB
		Motor GND	GND	PCB
Wheel Motor 3	Lower Deck, wires route to PCB	Motor 12V Input	12V Input	PCB

		Motor GND	GND	PCB
Wheel Motor 4	Lower Deck, wires route to PCB	Motor 12V Input	12V Input	PCB
		Motor GND	GND	PCB
LiPo Battery	Lower Deck, wires route to PCB	11.1V Output	Motor Voltage (VM)	PCB, Motor Driver 1
			Motor Voltage (VM)	PCB, Motor Driver 2
		GND	GND	PCB, Motor Driver 1
			GND	PCB, Motor Driver 2
USB Battery	Lower Deck, wires route to Pi	Output	USB Power Port	Raspberry Pi
		GND	USB Power Port	Raspberry Pi

APPENDICES

B. Bill of Materials

				Tota Line Price:	$ 215.19	Actual Price:	$ 180.87
System	**Part**	**Candidate Part**	**Quantity**	**Unit Price**	**Line Price**	**Fraction Used**	**Adjusted Price**
Computer							
	On-board PC	Raspberry Pi 4	1	N/A	$ -	1.0	$ -
	Microcontroller	Arduino Mega 2560	1	N/A	$ -	1.0	$ -
Drive System							
	Wheel DC Motor	Tsiny TS-25GA370H-45 Brushed DC Motor	4	N/A	$ -	1.0	$ -
	Motor Driver	SparkFun TB6612FNG	2	N/A	$ -	1.0	$ -
	Omni-Wheels	97mm Mecanum Wheel (set of 4)	1	$ 40.00	$ 40.00	1.0	$ 40.00
	Bracket	3D printed	4	N/A	$ -	1.0	$ -
	Shaft Adapter	3D printed	4	N/A	$ -	1.0	$ -
Sensor System							
	Camera	Pixy 2	1	N/A	$ -	1.0	$ -
	Range Sensor	HC-SR04 Ultrasonic Distance Sensor	2	N/A	$ -	1.0	$ -
Power							
	LiPo Battery	Turnigy 3s 11.1V 1300 mAh LiPo Battery	1	N/A	$ -	1.0	$ -
	XT60 Adapter	Standard XT60	1	N/A	$ -	1.0	$ -

	PCB	Custom PCB for motor drivers	1	$ 15.00	$ 15.00	1.0	$ 15.00
Chassis							
	Deck Material	Particle Board		N/A	$ -	1.0	$ -
	Fasteners	M3 Screw/Washer/Nut Kit	1	$ 12.99	$ 12.99	0.5	$ 6.50
	Extra Washers	#6 Flat Washers (pack of 30)	1	$ 1.15	$ 1.15	0.5	$ 0.58
	Corner Brackets	1-1/2 in Corner Brace (pack of 4)	2	$ 2.82	$ 5.64	1.0	$ 5.64
	Sheet Metal	6 in. x 18 in. 16-Gauge Plain Steel Sheet Metal	1	$ 10.35	$ 12.99	0.4	$ 5.20
	Square Tube	36 in. x 3/4 in. x 1/16 in. Aluminum Square Tube	2	$ 14.99	$ 29.98	0.8	$ 23.98
	Flat Bars	1 in x 36 in Aluminum Flat Bar 1/8" Thick	1	$ 7.74	$ 7.74	0.8	$ 6.19
Kicking							
	Spring		1	$ 10.00	$ 10.00	1.0	$ 10.00
	Pillow Blocks		2	$ 11.12	$ 22.24	1.0	$ 22.24
	Bolts for Blocks	1/4 in.-20 x 1-1/2 in.	1	$ 1.28	$ 1.28	1.0	$ 1.28
	Nuts for Blocks	1/4 in.-20 Zinc Plated Hex Nut	1	$ 1.28	$ 1.28	1.0	$ 1.28
	Sprocket		1	$ 17.69	$ 17.69	1.0	$ 17.69
	Collars		2	$ 5.44	$ 10.88	1.0	$ 10.88
	Rotary Shaft	3/8" x 12 in 6061 Aluminum Round Rod	1	$ 12.99	$ 12.99	0.5	$ 6.50
	Bike Chain		1	$ 10.84	$ 10.84	0.5	$ 5.42
	Hooks for Release	1/16 in. Zinc-Plated Wire Rope Clamp (3-Pack)	1	$ 2.50	$ 2.50	1.0	$ 2.50
	Loop for Release		1	N/A	$ -	1.0	$ -
	Striker	Size 3/4 Aluminum Pipe	1	N/A	$ -	1.0	$ -
	Servo	SpringRC SM-S4303R Continuous Rotation Servo	1	N/A	$ -	1.0	$ -