Fetcher Car for Ball Sports

ECE 445 Final Report

Project Team #34

Louie Davila, Yinshuo Feng, Patrick Sarad

TA: Sainath Barbhai
Professor: Viktor Gruev

Contents

INTRODUCTION

Purpose:

It is well known that athletes all around the world train for hours every day in order to hone their skills and become the best at their sport. Of course, no matter the training regime, athletes simply can't train all day and have to partition their time accordingly in order to maximize the gains they can make. As such, we can see here that time is a key factor when it comes to practicing. This is especially the case when people have to repeat a certain skill for hours in order to master it. Now, it is important to note that this isn't always the case. For instance, pro-athletes train for a living so they have more time to hone their skills. Younger athletes, on the other hand, often have an extremely tight schedule and have to fit in their training with school work or something similar. Our project aims to help these athletes in their training. More specifically, our project is designed to help athletes that train in ball sports.

In ball sports, many athletes can probably attest to both the annoyance and time-consumption of having to collect their ball after practicing a certain skill. For instance, in basketball, when practicing your shots, you may find that once you shoot all the balls that you had on standby, you have to go and pick them all up and bring them back to your initial training position. As another example we have tennis, where players have to go and pick up all of the tennis balls they used to practice their serves. The collection of these balls is a huge waste of time and, in many cases, takes more time than the actual act of practicing the skill. We intend to solve this problem by using a small, lightweight car with pincers that can watch for a ball from a safe position and fetch the ball when it leaves a designated region for practice. After fetching the ball, the car will return the ball to the user and will continue waiting for the next ball. This solution is the first of its kind in the sense that, with more time, manpower, and resources, the design could be expanded upon and refined for use in any ball-oriented sport.

Functionality:

1. The fetcher car can locate the ball, the user, or the waiting location as necessary in the current context of the program within 10 seconds of beginning a search.
2. The fetcher car will be able to run with the wheels at maximum power output for at least 45 minutes before needing to recharge.
3. The fetcher car will be able to capture and return a ball to the user within 45 seconds of the ball becoming acquirable. A ball is considered "acquirable" when it is within range of the ultrasonic sensors.

The first functionality is necessary because the image processing needs to be accurate and fast enough to notify the design of changes in the goal's position from several meters away and with enough time for the design to use said information while it is still relevant.

The second functionality is required because of how long we anticipate practice sessions to last for. If our car were to only run for 10 minutes, then it would provide no value to the user. With a 45 minute runtime, we hope that the project can find a balance between an affordable power source as well as a long enough runtime for the user to get adequate training in.

In order to solve our problem, the design needs to complete its objective fairly quickly. If it can't, then the user would have a more efficient practice session if they didn't use the design.

Subsystem overview:

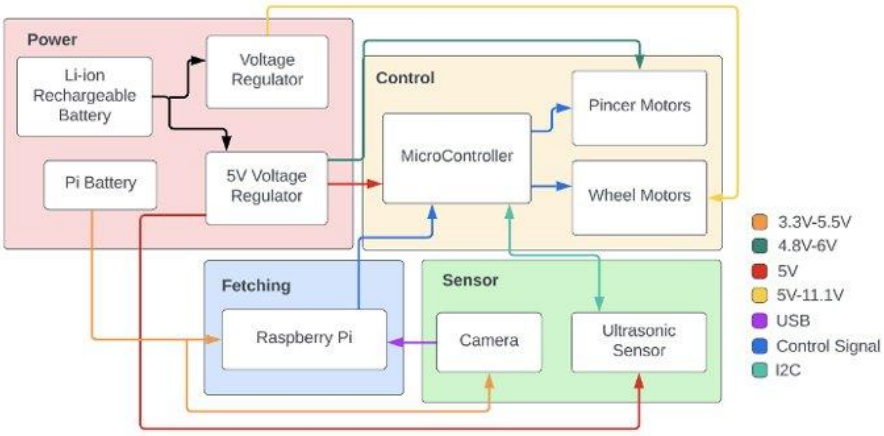


Figure 1: Block Diagram

Power:

The power subsystem supplies power to all other components on the board using 2 separate batteries and 2 DC-DC buck converters. One of the batteries goes to the Raspberry Pi and has no external regulation. This battery operates at 5V which is the required input voltage for the Pi. The other battery is connected directly to our PCB. The power from this battery has to pass through the 2 buck converters before it can be delivered to any of the components on the rest of the board. This is due to the power supply having a voltage output of 11.1V which is more than the 5V maximum that the rest of the components of the design can take in. One of the buck converters is connected to all of the DC motors on the car while the other buck converter is connected to all the other components. This separation is required as the DC motors draw the most power out of any component in our design. Since the buck converters have a maximum current output of 1A, we want to ensure that our design doesn't reach a point where the car can't operate due to insufficient power.

It is important to note that in our final product the power subsystem was altered. Instead of 2 separate batteries, we only used one. This change in design was required as the buck converters we used did not work with the PCB layout that we had. This single battery that we used was the one originally meant for just the Pi. Since this battery operated at 5V, we were able to directly connect it to the PCB without the buck converters (since the buck converters were there for the exact reason of producing an output of 5V).

Sensor:

The camera input will be passed through image processing, and the output will be two lists. The first list will have the duration of time (in seconds) that a goal has been detected in one of the 6 regions of the camera view without leaving that region. Every goal will never be detected in more than 1 region of the camera view for each frame. The second list will keep track of the last known region in which the goal was detected. This accounts for cases in which the goal is detected in the camera view, but later escapes detection. This could happen for several

reasons. For example, if the goal moves out of the camera view before the next frame can be sampled, or if the color tracking temporarily loses track of the goal due to issues with the lighting. In such cases, the design will behave as though the last frame in which the goal was detected is the current frame. The ultrasonic sensors will still be used to detect obstacles in this case.

3 ultrasonic sensors are included in the final design, with 2 on the front and one on the back. The 2 front sensors are used for detecting the ball within range of the pincers and detecting obstacles such as walls, respectively. These sensors are used only by the CHASE and ACQUIRE states. The back sensor is used for detecting obstacles such as walls in the FETCH and RETURN states. The obstacle detection proved to be a necessary feature due to the fetching subsystem's behavior of moving blindly when a goal that was in the camera view escapes the camera view, which could cause a collision.

Control:
The control subsystem interacts with all other subsystems in order to make the fetcher car move. The main processing unit is an ATmega328 microcontroller that takes digital inputs from the Raspberry Pi located in the fetching subsystem as well as takes digital measurements from the ultrasonic sensor located in the sensor subsystem. These digital signals are used to operate both the DC motors that are being used for the wheels as well as the servo motors being used for the pincers.

Fetching:
The fetching subsystem interacts with the control subsystem via the GPIO pins on the Pi, and takes as inputs the outputs of the image processing code. It will use these interfaces to define the behavior of the WAIT, CHASE, ACQUIRE, FETCH, and RETURN states of the FSM. Specifically, the image processing interface will be used to track the goal in all 5 states, and the control interface will be used to communicate positional data about each goal to the control subsystem in the CHASE, ACQUIRE, FETCH, and RETURN states. The goal is the ball (green) in the WAIT, CHASE, and ACQUIRE states, the user (red) in the FETCH state, and the waitpoint (blue) in the RETURN state.

DESIGN

Equations and Simulations

We implemented a runtime sampling feature that can force the FSM to simulate normal behavior in one specific state. While this feature is active, the program samples the time taken for the Pi to complete one full loop of the state function being simulated while counting the number of interrupts from the microcontroller that indicate the end of a loop. The program then computes the average of the recorded per-loop runtimes for the state being tested. This average is saved, and the average divided by the number of interrupts from the microcontroller is saved as the average per-loop runtime of the MCU code for the state being sampled. We have a script that executes the software with this feature enabled once on each state, where each execution yields 100 runtime samples.

Design Description and Justification

Power:

The power subsystem consists of two batteries that each power their own section of the design. One of these batteries powers the Raspberry Pi which is located in the fetching subsystem. It has an output voltage of 5V with a maximum current output of 3A. In addition to powering the Pi, this battery also serves the purpose of powering the two cameras that are connected to the USB ports of the Pi. The second battery in our design is used to power all components located on the PCB. This includes the microcontroller, the DC motors, the servo motors, and the ultrasonic sensors. All of the components listed, aside from the DC motors, can only be powered with a maximum of 5V. Since the battery to the PCB has an output voltage of 11.1V, we had to also incorporate two DC-DC buck converters that regulated the voltage down to 5V. It's important to note that these buck converters have a maximum current output of 1A. When under a high load, the DC motors are capable of pulling around 700-900mA of current. As such, if we were to use only one buck converter, we could potentially exceed the amount of current that the converter could supply, resulting in the converter overloading and shutting down. With two converters, we have the potential to supply up to 2A of current which is more than enough to keep the design running safely.

As described above, we used two separate batteries to power our design. Instead of doing this, we could have simplified our design and used one battery for the whole project. The reason why we didn't do this was because we already had a battery for the Pi that was specifically designed for it. This meant that we had fewer components to power and therefore a wider range of batteries to choose from.

In our final implementation of the project, we actually only used the Pi battery to power everything. We did so by connecting the PCB to the 5V and GND pins of the Pi. Now, ideally we would follow the procedure described above when powering all of the components in our project as it would ensure maximum current could get delivered to the whole design and would also minimize the strain on each individual battery. However, due to a lack of understanding of the datasheets for the buck converters, we were not able to use the buck converters with the PCB design that we had. This was due to us not using the "remote control" pin of the buck converters properly (they were always off as a result). So, the best alternative we had was to use the Pi battery to power the rest of the components in place of the buck converters and 11.1V battery since the Pi battery had a voltage output that perfectly matched the required voltage inputs of all the other components.

Sensor:

We used 3 HC SR04 ultrasonic sensors in our design, and a Logitech C925e and a Logitech C930e USB Webcam in our design. One ultrasonic sensor is mounted on the ball pincers, connected to the microcontroller, and used to detect the distance to the ball. Another ultrasonic sensor is mounted on the front side, connected to Raspberry Pi GPIO pins to determine the distance to obstacles. A third ultrasonic sensor is mounted on the rear side of the car to detect distance to the obstacles when reversing.

We use the Logitech C925e for the front camera because it has a narrower 78 degrees field of view. This allows us to have better focus when looking for the small tennis ball. We use the Logitech C930e for the rear camera and it has a wider 90 degrees field of view. Both are connected to the Raspberry Pi via USB ports.

Control:
The control subsystem has the job of making both the car and the pincers on the car move in accordance with whichever state we are located in. In order to fulfill this job, the microcontroller has to take in digital signals from both the Pi and the ultrasonic sensors and process these signals in order to determine how to operate each motor.

The microcontroller takes in 8 inputs from the Pi: two left motor controls, two right motor controls, two pincer controls, a mode control, and an interrupt. The left and right controls are used to determine which direction the DC motors should spin. In order to control the direction of these DC motors we used h-bridges. The left and right controls are essentially transformed into a 2 bit code that the h-bridges use to determine if the motors should spin forwards, backwards, or if they should stop. Integrating the h-bridges into our design was necessary as we needed a way to control the direction of motion of the motors. Without h-bridges, we would be limited to only forward motion with the motors. This type of limited functionality is undesirable as it would mean that our car would be stuck in the case that it runs into a wall or something of the sort. The pincer controls were used to operate the servo motors that moved the pincers. One bit of the pincer controls was used to power the servos on and off. Since we only need the servo motors to be on when we are catching the ball, we power them off in all other instances so that we drain less power from our battery. The second bit of the pincer controls is used to determine whether the pincers should open or close (open when chasing the ball and close when the ball is right in front of the car). The mode control was used for testing. Enabling the mode control bit meant that we had manual control of all of the motors in our design. This was used to debug each individual signal sent by the Pi. It was also used to test the speed of the pincers and the DC motors. The final control signal sent by the Pi was an interrupt bit. Since reading in signals can take a lot of time, we wanted to make sure that our design only attempted to read a signal when a signal was actually being sent.

Measurements were always being read from the ultrasonic sensors. These measurements were used to determine whether the ball was close enough to the car to engage the pincers. Since these measurements only took a few milliseconds maximum to run, there was no need to optimize when they were taken. Additionally, we wanted the constant readings from the sensors in order to ensure that the car wasn't about to hit a wall or some other object.

Fetching:
We chose to implement the fetching subsystem as a simple FSM in order to reserve enough hardware resources for the image processing and control subsystem interfaces. This especially paid off in our final design, where even a minor increase in the software complexity of the design

was enough to drop the FPS of the image processing to 1 FPS or less on average. For information about the states and transitions of the FSM, refer to Figure 7.2 on page 12

The FSM has been structured this way so that the WAIT, CHASE, ACQUIRE, and FETCH states each represent a key action that must be performed in order to retrieve a ball for a user. That is, the design must detect a ball in the WAIT state, travel to it in the CHASE state, grab it in the ACQUIRE state, and travel back to the user and release the ball in the FETCH state. The RETURN state ensures that the design can restart at the WAIT state with the same view as it had before it started chasing the ball. This is important because it eliminates the need for the user to manually intervene and reposition the design so that it can see a dropped ball. If the user had to do this every time that the design retrieved a ball for the user, it would defeat the purpose of our design. The transition from CHASE to RETURN accounts for the possibility that a ball was detected, but the design cannot find a path to it. This could occur if, for example, a ball were to bounce off of a court and into the stands. The transition from ACQUIRE to RETURN is a means of determining when the design has failed to meet the 45-second time limit to capture an acquirable ball, as specified in our high-level requirements.

The CHASE, ACQUIRE, FETCH, and RETURN states all use the bottom-center region for transitions because it is assumed that the design is operating on a flat terrain without extra obstacles, meaning that any goal in the bottom-center region can be reached by slightly adjusting the design's orientation and traveling in a straight line. This is used in transition 2 to prevent the design from colliding with the ball and pushing it away, in transition 4 to prevent a collision with the user and allow the design to move close enough to the user, and in transition 5 to prevent a possible collision with a wall that may be near the waitpoint. Transition 3 uses this region along with the ultrasonic sensor proximity data to determine when a ball is within range to be grabbed by the pincers. Transition 3 is intended to be the most difficult transition for the design to execute, because capturing the ball is the most physically complex action performed by the design. Also, the design has no way of confirming that the ball has been successfully captured once the pincers have closed, so the design must automatically start looking for the user after closing the pincers, even if the ball wasn't captured. In this way, the success of the design is dependent upon the precision of the conditions for satisfaction of transition 3. Unfortunately, the ultrasonic sensors proved to be a very imprecise option for measuring distance. So all states dependent upon the ultrasonic sensors couldn't be fully tested for accuracy. For the final demo, all inputs to the fetching subsystem regarding the ultrasonic sensor data were set to only return a value within the transition range, meaning that they were practically not included in the final design. Otherwise, the design behaves as expected in each state.

Subsystem Diagrams and Schematics

Power:

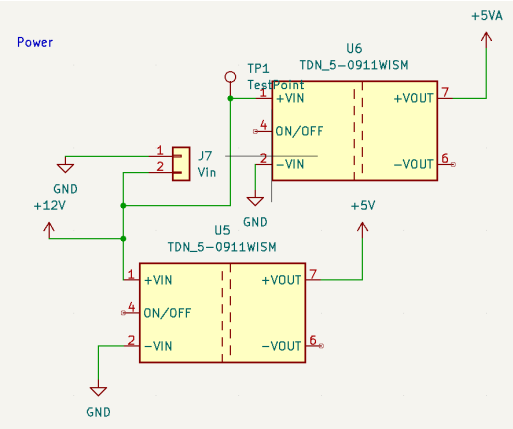


Figure 2: Power Subsystem Schematic

Figure 2 displays the power subsystem design located on the PCB. “Vin” is where the battery is meant to output power. Both rectangles named as “TDN_5-0911WISM” are the buck converters used for the design. They are identical and differ only in where their outputs are going. The top converter has its output going to the DC motors while the bottom converter has its outputs going to the rest of the components on the PCB.

Sensor:

For determining the distance to a normal object, the program will simply send the trigger signal to the ultrasonic sensor, read the pulse in time, and divide the reading by a constant number. For determining the distance to the tennis ball, because the ball absorbs sound and thus has a strange sensor reading pattern, the program will store the last 20 readings, run a loop over it to check if there is a reading smaller than 10 and another reading reading larger than 2000, return true if both conditions are met.

The image processing code will run in a loop, read one frame from the camera in each iteration, apply the corresponding colormask to the image, and find the maximum contour in the filtered image. The center of the maximum contour is calculated and returned to the current state of the FSM.

Control:

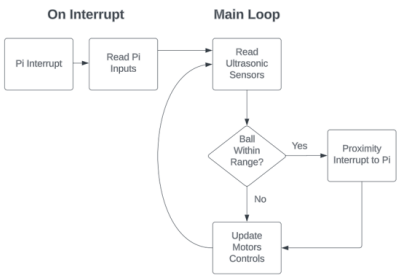


Figure 3: Flow Chart for Control Subsystem Logic

Figure 3 displays the basic flow chart of the microcontroller. In its main loop it simply reads ultrasonic data and updates all motor controls accordingly. At any point during its runtime, if the Pi sends an interrupt signal, the main loop of the microcontroller pauses and the control signals are read in. Once all signals are updated, the microcontroller continues its main loop operation.

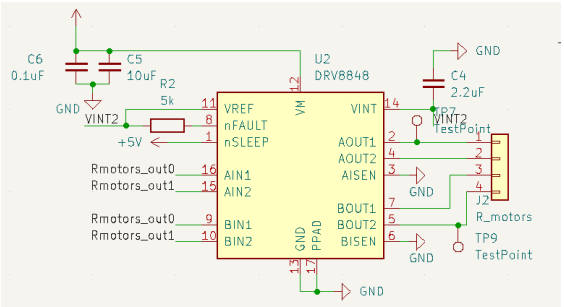


Figure 4: H-Bridge Schematic

Figure 4 displays the schematic of one of the h-bridges used in the design. All other h-bridges used follow the exact same setup. The h-bridges take in as input 2 control signals (denoted as Rmotors_out0 and Rmotors_out1 in the image above). These control signals determine the flow of current through the DC motors which are connected to the block called “J2 - R_motors”.

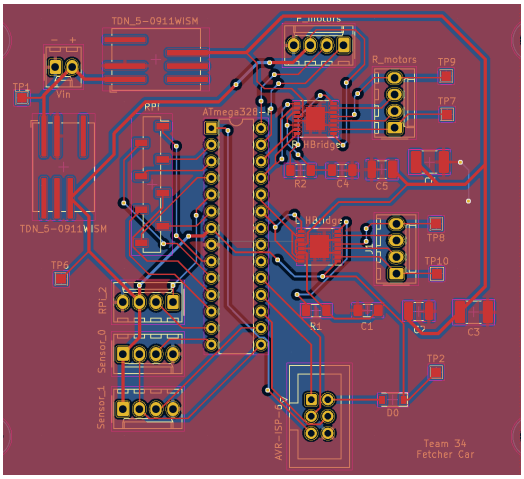


Figure 5: Entire PCB layout

Figure 5 displays the design of the whole PCB design. Apart from the top left section of the image, the rest of the components displayed on the schematic are used in the control subsystem. Inputs to the PCB come from the blocks called “RPi”, “RPi_2”, “Sensor_0”, and “Sensor_1”. The microcontroller is located in the center (called ATmega328) and all outputs to the motors are located on the right (called “P_motors”, “R_motors”, and “J1”).

Fetching:

	Pin # (Pi)	I/O	Pin #	I/O
1	17	O	9 (ATmega328P)	I
2	27	O	10 (ATmega328P)	I
3	12	O	2 (ATmega328P)	I
4	13	O	13 (ATmega328P)	I
5	4	O	11 (ATmega328P)	I
6	25	O	12 (ATmega328P)	I
7	5	O	6 (ATmega328P)	I
8	6	O	4 (ATmega328P)	I
9	16	I	5 (ATmega328P)	O
10	26	I	16 (ATmega328P)	O
11	18	O	2 (front-sensor)	I
12	24	I	3 (front-sensor)	O
13	20	O	2 (back-sensor)	I
14	21	I	3 (back-sensor)	O

Figure 6: GPIO Pin Assignments

Rows 1-8 are the Lmotors_in0 (17→9), Lmotors_in1 (27→10), Rmotors_in0 (12→2), Rmotors_in1 (13→13), Pincer_on (4→11), Pincer_direction (25→12), CTRL (5→6), and Pi_INT (6→4) bits sent from the Pi to the microcontroller. Lmotors_in0, Rmotors_in0, and Pincer_on all tell the microcontroller whether or not it should be using the right or left side car motors and the pincer motors (1 if yes, 0 if no). Lmotors_in1 and Rmotors_in1 tell whether to move the car motors on the respective sides forward (0) or backward (1), and Pincer_direction tells whether to open (0) or close (1) the pincers.

Note that row 4 (Rmotors_in1) and row 10 (Proximity_Data) both occupy different pins on the MCU than what was proposed in the original design. This is because we encountered an issue with pin 3 (PD1) on the MCU wherein the pin was constantly high regardless of the input from the Pi. In the PCB design, pin 16 (PB2) on the MCU was reserved for the response from the second ultrasonic sensor when the MCU triggered the ultrasonic sensors. This pin was repurposed as the Proximity_Data output from the MCU to the Pi. Pin 13 (PD7) on the MCU was reserved for the Proximity_Data output from the MCU to the Pi. This pin was repurposed as the Rmotors_in1 output from the Pi to the MCU.

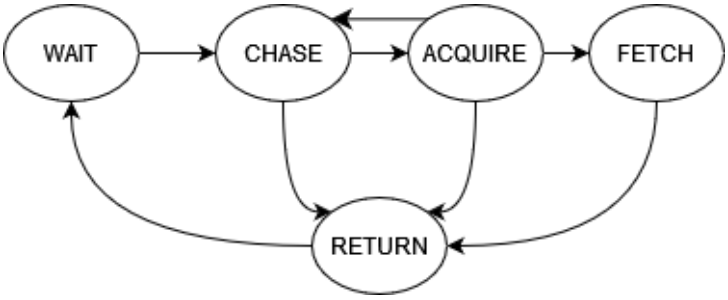


Figure 7.1: Old FSM Diagram

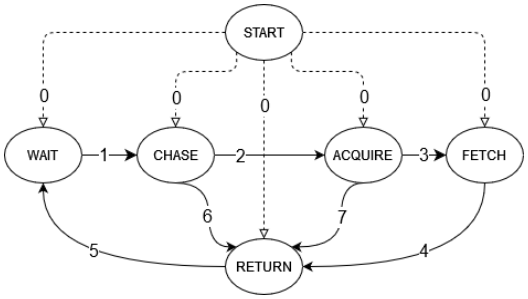


Figure 7.2: New FSM Diagram

The fetching subsystem is implemented as an FSM with 5 states and 1 initial state. The state transitions are defined as follows:

0. START → {WAIT, CHASE, ACQUIRE, FETCH, RETURN}: Occurs only once when the FSM is initialized
1. WAIT → CHASE. Occurs when a ball has occupied the same region for 2 seconds. This transition sets the 60-second timer for the CHASE state.
2. CHASE → ACQUIRE: Occurs when the ball being chased is detected in the bottom-center region of the screen, and it sets the 45-second timer for the ACQUIRE state
3. ACQUIRE → FETCH. Occurs when the ball is detected in the bottom-center region of the camera view and the ultrasonic sensors detect an object in grabbing range of the pincers.
4. FETCH → RETURN. Occurs when the user is in the bottom-center region of the camera view and the rear ultrasonic sensor code reports an object within 50cm.
5. RETURN → WAIT. Occurs when the waitpoint is detected in the bottom-center region of the camera view and the front ultrasonic sensor code reports an object within 50cm.
6. CHASE → RETURN. Occurs if the FSM spends more than 60 consecutive seconds in the CHASE state. The attempt to reach the ball has failed.
7. ACQUIRE → RETURN. Occurs if the FSM spends more than 45 consecutive seconds in the ACQUIRE state. The attempt to acquire the ball has failed.

As shown in Figure 8.1, the old FSM included a transition from ACQUIRE to CHASE. This transition was added to handle cases where the ball is not detected in ACQUIRE for some predefined amount of time. In such a case, we would be able to re-enter CHASE to focus on finding the ball. This case was removed because it inhibited our ability to verify that the design could capture an acquirable ball within 45 seconds, as asserted in our high-level requirements. In its place, some extra behaviors from CHASE were added to ACQUIRE so that the design could search for the ball changing states.

Figure 8.1: Complete Pi Software Flowchart Software

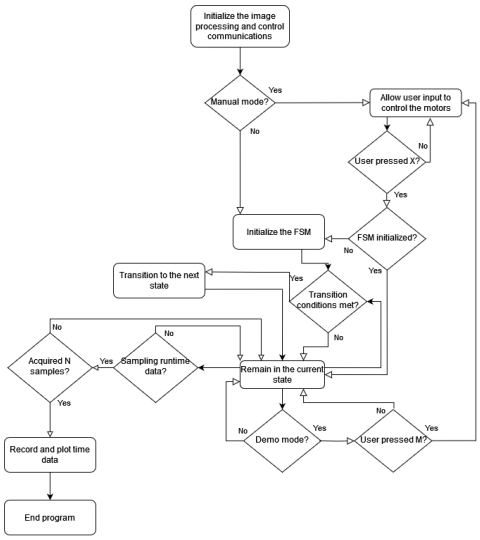
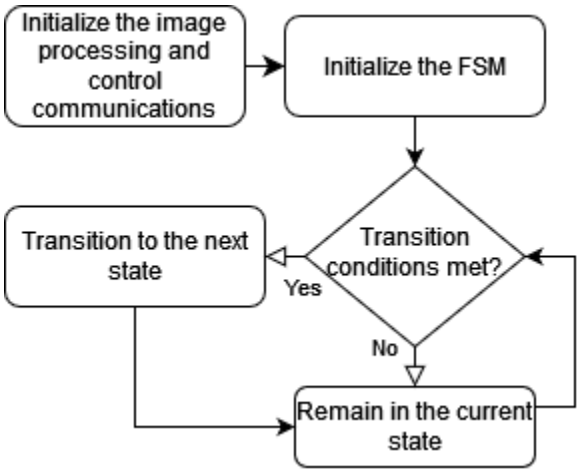


Figure 8.2: Pi FSM



COST AND SCHEDULE

Cost:

We found that the typical post-graduate salary of an ECE student at UIUC is about $90,000. Based on this value, the hourly salary we would receive would be about $44. Each individual on our team would make $44/hour x 2.5 x 100 (hours to complete) = $11,000. The total labor cost would then be 3 x $11,000 = $33,000 for all 3 group members.

For the parts we used, the total cost comes out to $325.57. Individual prices of each component are displayed in figure 9. Assuming we add additional costs from 5% shipping tax and 10% sales tax, we get a new total of $374.40. Adding up labor costs and part costs, our final project cost would be $33,374.40.

Items	**Cost**
Logitech C925e USB Webcam	$0 (borrowed)
Logitech C930e USB Webcam	$130
3 HC SR04 Ultrasonic Sensor	$13.50
Raspberry Pi 4 Model B	$75

DC/DC Buck Converter	$55
5k ohm Resistors	$4.92
10uf Capacitors	$3.27
0.1uf Capacitors	$3.36
2.2uf Capacitors	$2.07
ATmega328	$2.63
DRV8848 H-Bridges	$5.82
PiSugar S Plus	$30

Figure 9: Table displaying costs of all components used

Schedule:
The link below is a Gantt chart that displays the weekly schedule we followed as well as what each individual worked on.
Link:
https://docs.google.com/spreadsheets/d/11F2bXZBZZVbTI17Cs0ifYs-3oXvgZa6a/edit?usp=sharing&ouid=103313886692512844099&rtpof=true&sd=true

REQUIREMENTS AND VERIFICATIONS

Completeness of Requirements:

Power:

The power subsystem is meant to provide enough power to the design so that it can run for at least 45 minutes at full capacity. In our final product we only used a single battery rated at 5V and 5Ah. This gave us a total of 25Wh to work with. All parts located on our PCB had a maximum current draw of 1.5-1.68A at around 4.5-4.9V. All of the relevant aforementioned information regarding the power supplied to the PCB can be found in figure 10. The Pi and the cameras attached to the Pi had a maximum current draw of 3A at 5V. So, the project as a whole consumed around 21.75-22.35W of power at a maximum. This means that, with the given power supply, we are able to run our car for a minimum of 1.1 hours. We can see here that the 45 minute requirement is met.

Since the PCB components only need a max combined current of 1.68A to function, we can say with certainty that everything will be powered without any issues. We also know this to be the case as the design was able to pass some field testing when we powered it on with the Pi battery.

Sensor:

The requirements for the ultrasonic sensors are not fully met due to the unexpected problem of the ball absorbing the sound. The original requirements need the sensor to generate a

consistently reliable distance reading from 1 meters to 5 centimeters which is the minimum range for the sensor. Instead we have achieved a constant 80 centimeters reading from 25cm to 40cm, and inconsistent varying readings in the 20 centimeters range. The alternative spike detection methods could also determine the ball existence from 20 centimeters to 10 centimeters range.

The requirements for the cameras are not fully met due to the change of camera hardware and the tiny size of the ball. The original requirements need the camera being able to identify the ball up to 7 meters with at least 2 frames per second. With the lowered resolution from 4000 * 3000 pixels to 1920 * 1080 pixels, as well as the inability to use detection methods other than maximum color contour, the detection range is limited to about 3 meters in a light sufficient and clear environment. In terms of frame rate, we achieved a higher frame rate to at least 3 frames per second.

Control:
All control requirements were fulfilled. First off, we were able to confirm that the microcontroller was effectively communicating with the h-bridges and was driving the motors in the correct direction by manually sending inputs from the Pi to the microcontroller. These inputs simply moved the wheels and the pincers in different directions without the use of any image processing.

We were also able to ensure that the microcontroller code executed faster than 50ms. We did so by measuring how many loops of the microcontroller ran during the execution of a loop of a state function in the FSM. By taking the average time the state took to run and dividing it by the amount of loops completed by the microcontroller, we got the average time taken by the microcontroller to finish a single loop. With this method, we found that a single loop took less than a millisecond on average to complete, which is way faster than the 50ms threshold we placed.

Fetching:
The only requirement specific to the fetching subsystem was that it executed within 500ms, with an expected minimum runtime of 100ms. Using the runtime sampling feature described in “Equations and Simulations”, we sampled the runtime under two conditions. The first condition simulates the case in which the motors are drawing the maximum expected current of about 1.7A, and the results can be seen in figure 14 on page 18 The second condition simulates the case in which the motors are drawing almost no current in each state. The results can be seen in figure 15, and it was under this condition that we sampled the average runtime of the MCU code. Note that the CHASE, ACQUIRE, FETCH and RETURN runtime averages change by notably different amounts between the two figures. This is due to the fact that changes in lighting between the test environments of conditions 1 and 2 caused the motors to move less for certain states in condition 1 and more for certain states in condition 2. However, we can use the similarity of the CHASE, ACQUIRE, FETCH, and RETURN states to generalize the maximum runtime of 315ms as the overall maximum runtime for each of the 4 states. Since WAIT never uses the GPIO pins or does any extra work compared to the other states, its runtime can be

used to estimate the minimum expected runtime of all 5 state functions. We can compare the changes in the WAIT runtimes between the 2 conditions in order to get a minimum expected runtime of roughly 230ms for each state. Since the range of 230ms-315ms is within the required range of 100ms-500ms, the requirements for the fetching subsystem are satisfied.

Quantitative results:

Power:

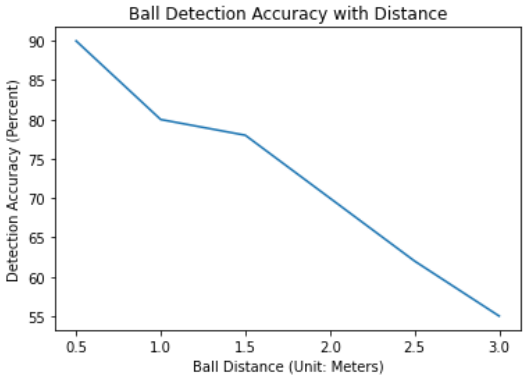
	Current	Voltage
Full PCB	0.450-0.510A	4.5-4.9V
Full PCB Not Active	0.036-0.038A	4.9V
Full PCB High Load	1.5-1.68A	4.5-4.9V

Figure 10: Voltage and Current Measurements of PCB components

Figure 10 displays the current draw and voltage levels of the PCB at different operation levels. “Full PCB” describes everything that was attached to the PCB: ultrasonic sensor, DC motors, servo motors, and the microcontroller. Since we couldn’t individually measure the current draw of each component, we measured the total current draw of all the components combined. We used these measurements to calculate the power we would need to supply in order to run our design.

Sensor:

The FPS of the cameras reflects the maximum and minimum runtimes of the state functions as shown in the bar plots for the fetching subsystem runtimes. So, the FPS can be expected to stay in the range of 3 to 5 FPS.



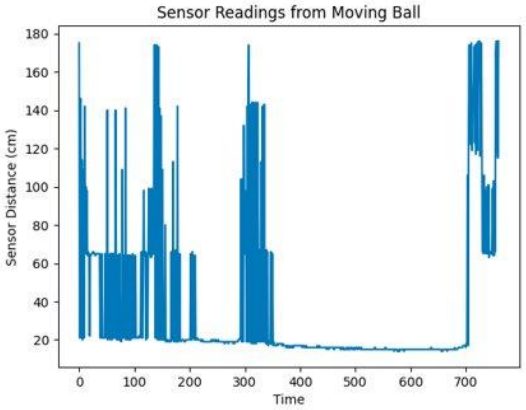


Figure 11: Ultrasonic Sensor Distance Measurements

The graph in figure 11 displays sensor readings as a function of time. The readings are taken as a tennis ball is moved closer and closer to an ultrasonic sensor. As can be seen in the image, there is a lot of noise in many parts of the graph. This unexpected behavior is why we failed to incorporate the ultrasonic sensors into the design.

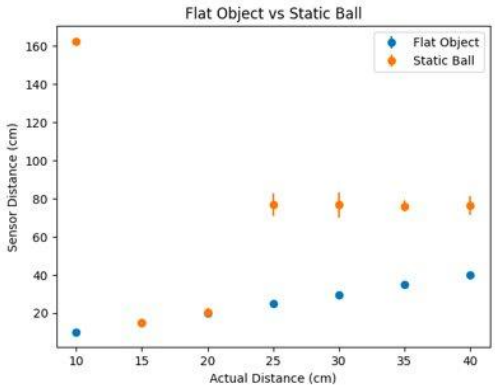


Figure 12: Static Distance Measurements of Different Objects

In figure 12 we have a graph of flat object distance compared to tennis ball distance. As can be seen, a flat object is more accurately measured and has measurements that are less varied. A tennis ball, on the other hand, has measurements that can vary greatly and can deviate from the expected path.

Control:

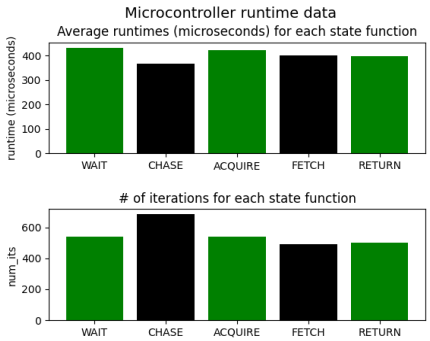


Figure 13: Microcontroller Runtime

Figure 13 displays the runtime data of the code that is executed on the microcontroller. In order to maximize the framerate of the cameras we were using, we had to make sure that the code didn't take any longer than 50ms to complete. As can be seen, the max time taken at any point is 400us, which is far less than the maximum threshold.

Fetching:

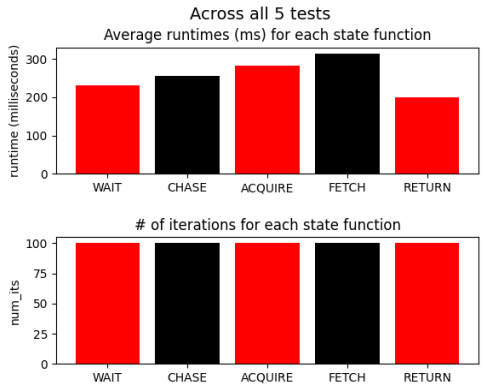


Figure 14: Runtime Data for Each State of the FSM (condition 1)

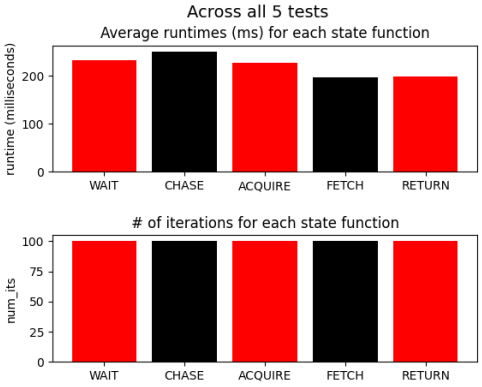


Figure 15: Runtime Data for Each State of State of the FSM (condition 2)

Figure 14 shows the average runtimes for the design when the motors are drawing about 1A of current. In this case, the runtimes are slowed down because the Pi battery is struggling to supply enough power to the whole design. The maximum expected runtime under this condition is about 315ms. Figure 15 shows the average runtimes for the design consuming 0.036-0.038A, in which case the battery has no problem supplying the required 15W to the Pi. The minimum expected runtime under this condition is about 230ms.

CONCLUSION

Accomplishments:

The fetching and control subsystems, as well as the image processing from the sensor subsystem, were all successfully integrated into the design. As a result, our design was able to move the motors based on the regions of the camera view in which the image processing detected a goal. These movements reflected the design's purpose to move toward the objective, as the design would turn left when the objective is detected or was last detected on the left side of the camera view, right for the right side of the camera view, and forward for the center or top of the camera view.

Future work / alternatives:

An AI alternative to the fetching subsystem would most likely be a more effective approach than the FSM-based approach taken in this implementation. However, the Pi would need to either be replaced or be run in parallel with another device. An example of a good replacement would be the Jetson Xavier, as it has been observed running the same image processing code that was used in this implementation code at 10 FPS as opposed to the maximum 5 FPS on the Pi. Ideally, a parallel device for AI would have at least the same hardware capabilities as the Pi used in this design in order to balance hosting a sufficiently complex AI and communicating with

both the microcontroller and the Pi being used for image processing. This wouldn't only allow the fetching subsystem to be more dynamic, but it would also free up resources on the original device which can be used to implement more advanced techniques for image processing. An example of such a technique is hybrid detection, which would use a combination of color and shape detection.

Our design didn't include a means of verifying that a ball had been captured once the pincers had closed. This placed a disproportionately large responsibility for the success of the design on the precision/reliability of the transition condition from ACQUIRE to FETCH. This design flaw could have been avoided by including a sensor with the precise purpose of verifying whether or not an object has been captured by the pincers. Since the ultrasonic sensors proved themselves to be unreliable, alternatives like Lidar sensors may be a better choice. It might also be better to replace all of the existing ultrasonic sensors in the design with Lidar sensors in order to eliminate the issue of the ball absorbing sound.

Since the ultrasonic sensors failed, the design could only move towards the goals until they were detected in the bottom-center region of the screen. If we had used Lidar sensors instead, we could've had more samples to work with to verify accuracy of readings in a shorter time frame. This also would have avoided the issue of sound absorption by the tennis ball.

Ethical Considerations:
In the interest of security, it would be necessary to remove the time sampling feature from the design if it were ever to be made available as a product. The function responsible for keeping track of time data keeps several global variables, some of which are buffers that have no limit on length. Since this function is called during every SIGUSR1 interrupt, an attacker could potentially insert code of any length in the buffer and have the code executed with raised privileges by executing another interrupt.

Additionally, debugging features such as writing to log files or printing to the terminal during execution would have to be removed for security reasons. If not, then an attacker may be able to use a print formatting attack to execute code on the stack or a buffer overflow to make the code write anything that the attacker wants to any file in the filesystem. These could also be used by the attacker to expose the internal state of the design and inspect it for vulnerabilities that we overlooked.

Overall, we would need to review the entire code base carefully, several times, and do a great deal of security research in order to minimize the risk of breaching the customer's security due to overlooked vulnerabilities. Ideally, we would hire a security analyst or a team of security analysts to review our design and identify vulnerabilities.

Citations

[1] Rui Santos, "Complete Guide for Ultrasonic Sensor HC-SR04," Random Nerd Tutorials, 2018. [Online]. Available: https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. [Accessed: Mar. 28, 2023]

[2] "ATmega48A/PA, ATmega88A/PA, ATmega168A/PA, ATmega328/P Datasheet," Mouser Electronics, 2017. [Online]. Available: https://www.mouser.com/datasheet/2/268/ATmega48A_PA_88A_PA_168A_PA_328_P_DS_DS40002061B-3050139.pdf. [Accessed: Mar. 28, 2023].

[3] Tenergy Corporation, "AT Tenergy Li-Ion 18650 11.1V 5200mAh Rechargeable Battery Pack w/PCB (3S2P, 57.72Wh, 9A Rate)," Tenergy Power, 2019. [Online]. Available: https://power.tenergy.com/at-tenergy-li-ion-18650-11-1v-5200mah-rechargeable-battery-pack-w-pcb-3s2p-57-72wh-9a-rate/. [Accessed: Mar. 28, 2023].

[4] "pystatemachine - PyPI," Python Package Index, 2022. [Online]. Available: https://pypi.org/project/pystatemachine/#:~:text=pystatemachine%20is%20a%20versatile%2C%20yet%20easy-to-use%20finite-state%20machine,another%20when%20initiated%20by%20a%20triggering%20event.%20Usage. [Accessed: Mar. 28, 2023].

[5] "ACM Code of Ethics and Professional Conduct," Association for Computing Machinery (ACM), 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: Mar. 28, 2023]

[6] A. Rosebrock, “Ball tracking with opencv,” PyImageSearch, 17-Apr-2021. [Online].Available: https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/. [Accessed:29-Mar-2023].

[7] “Changing colorspaces,” OpenCV. [Online]. Available: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html. [Accessed: 29-Mar-2023].

[8] How to Mechanics. “Ultrasonic Sensor HC-SR04 and Arduino – Complete Guide.” Available: https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/ [Accessed: 14-March-2023]

[9] Arduino. “Servo.” Available: https://www.arduino.cc/reference/en/libraries/servo/ [Accessed: 21-March-2023]

[10] “DRV8848 Dual H-Bridge Motor Driver,” Texas Instruments, 2015. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8848.pdf?ts=1681218947815&ref_url=https%253A%252F%252Fwww.google.com%252F [Accessed: 9-Mar-2023]

[11] Institute of Electrical and Electronics Engineers. “IEEE Code of Ethics.”
Available: https://www.ieee.org/about/corporate/governance/p7-8.htm [Accessed: 20-Feb-2023]