Capstone Design Proposal

18-500 ECE Design Experience

Team C1

Rohit Pillai, David Yu, Ben Chen

March 8, 2018

Tennis Ball Retrieval System

Table of Contents

Tennis Ball Retrieval System 1

Table of Contents 2

Project Description 3

Design Requirements 4

Functional Architecture 5

Design Trade Studies 7

System Description 9

Evaluation 14

Project Management 16

- Schedule 16
- Team Responsibilities 16
- Budget 17
- Risk Management 18

Related Works 20

References 22

Project Description

Recreational tennis players spend a significant amount of time picking up balls instead of playing tennis. These balls are often scattered and far apart from each other, requiring players to frequently walk to and from the corners of the court. This wastes time and energy that the player could be using to rehydrate or rest. To address this issue, we built a robot that autonomously locates and picks up tennis balls. Once activated, the robot begins identifying the balls on the court and their respective locations; the bot then travels to them, picks them up, and stores them on its back. Ball detection is done using a combination of color thresholding and Canny edge detection. Localization is performed by using colored markers that designate the corners of the court. In order to make the product attractive to tennis players, the robot moves faster than human walking speed, ensuring that time and energy are both saved.

Design Requirements

Video feed:

To ensure that we have a responsive system, we require that our robot processes at least **2** frames per second. In addition, we need a minimum resolution of **256 x 144** so that there exists enough visual information to locate balls and markers. Larger resolutions would ideally result in increased range and accuracy.

Battery:

As our batteries drain from usage, the voltage supplied to the components drops as well. In addition, from research and testing, we have found that the voltage drop is not linear. In order to have consistent behavior with our motors, we require that the LiPo and NiMH batteries, the ones that power the servo/electric speed controls, be charged to full after 10 minutes of continuous use.

Environment:

We require that the surface of the playing field be of a non-rough, solid type, such as hardwood or smooth concrete. This ensures that the car does not run into any unexpected interference and that movement and turning is consistent. The area of operation has to be flat and large enough, at least 7x7 feet, in order for the car to be able to safely make all its movements. It is hard to quantify the amount of light required, but there needs to be enough for the camera to clearly discern shapes and colors.

Balls and markers:

The tennis balls are required to be of the yellow-green variety. This is the standard color of a normal tennis ball. Our vision should be able to detect variations in the shade of that color but the system will not look for balls of other colors. The markers used to indicate the corners of the field of play for our localization method are required to be bright and unique colors.

Functional Architecture

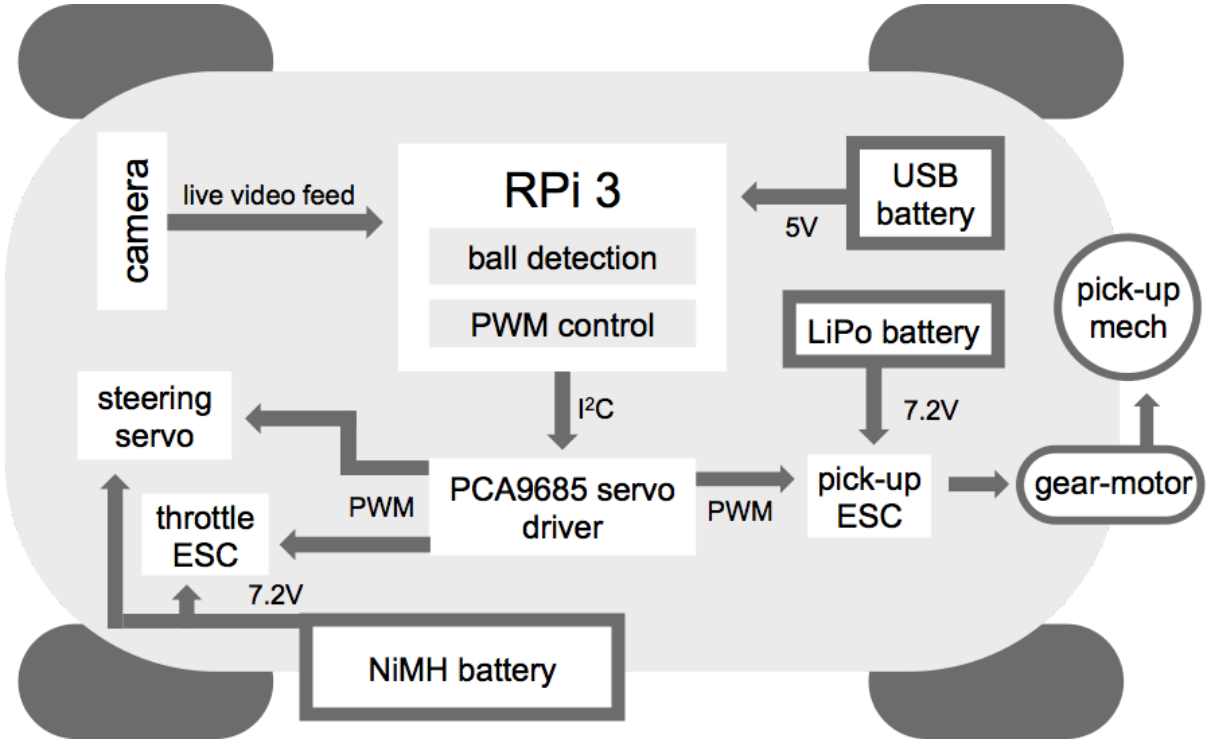

Figure A - a diagram of our system, which is contained on an RC car

Figure A shows the high level architecture of our system, and as per the diagram, each component rests on or is connected to a RC car. On top of our system sits the wide-angle lens **camera** that sends a live feed to our **Raspberry Pi 3,** which is powered by the 5V **USB battery**. Using computer vision, the Pi determines the behavior of both the car movement and the ball pickup, using PWM signals to control both. The PWM signals are sent through the I2C pins of the Pi to the **PCA9685 servo driver** which in turn sends out the PWM signals to each of the servos/electric speed controls. Two signals go to the **steering servo** (car steering control) and **throttle ESC** (car throttle control), which are powered by an **NiMH battery**; all three of these components are originally part of the RC car. A different PWM signal is also sent to the **pick-up ESC**, which controls the 75:1 high-powered **gear-motor** that runs the the **pick-up mechanism**; these are powered by a 7.2V **LiPo battery**. These are the interactions between the principle components of our architecture.

Design Trade Studies

Computer

We chose a Raspberry Pi 3 because of its widespread adoption and amazing price-to-performance ratio. Other options were the Nvidia Jetson TK1 and the PandaBoard, but those were much more expensive and we didn't believe we would be making the most of their additional features. Also, the Raspberry Pi platform had much more support.

Vehicle Platform

We toyed with the idea of building our own robot to move around with, but quickly realized that it was more effort than it was worth. After looking around at previous capstones and asking for feedback, we narrowed our choices down to RC cars and robot kits. However, most of the robot kits we found seemed too weak and slow for our purposes, so we decided to go with an RC car. Research showed that a 1/10 RC car was the most common size of hobby RC car, and therefore would be our safest choice because of the abundance of spare parts online. In addition, hobby RC cars all operated under similar protocols, which meant that there was a lot of useful documentation about programming them online. Further investigation revealed that the *Exceed* brand of RC cars was the most inexpensive option for a hobby RC car that was still reliable and popular. We ended up buying an Exceed Truck RC Car for its suspension capabilities, speed, reliability, and size.

Camera

At first, there were multiple candidates for visual input to our system. 360 degree cameras, dashcams, and even GoPros were considered. Aside from cheaper dashcams, however, these options were extremely expensive, and there were no clear ways to access their video streams in real-time on our Raspberry Pi. They all also required a separate power source. Our team was recommended the official Raspberry Pi Camera Module, which we found appealing due to its price and software support. There were some concerns over the viewing-angle, but we found a SainSmart wide-angle camera that was very similar to the official camera module, and so we opted for that after reading reviews of it. Both camera modules were directly powered by the Pi, and they could also stream video at 1920x1080 and 30 frames per second, which is overkill for the amount of compute power we have.

Software Language

Although C++ is heralded as the language of choice for pure performance, we opted to code our software side in Python. The OpenCV Python port is reported to be within 4% of performance with the OpenCV C++ equivalent, and Python is much faster to develop for. In addition, the picamera library supports video streams much simpler than the RaspiCam C++ library, and our team has more experience with Python than C++.

Pickup Mechanism

There were several approaches that we considered for picking up tennis balls. There was the tennis basket method of applying force from the basket's metal grid, compressing the ball, and having it pop up into the hopper. This was immediately ruled out as it would require an unreasonable amount of force. Another method we examined also required slight compression of the tennis ball. This approach used a rolling cylinder to compress the ball against a curved wall then use friction to guide a ball up to the container. Although requiring less force than the previous way, we still determined that our system would be unable to feasibly achieve this. One ball picking robot we studied used two fast spinning brushes to shoot the ball up and into the basket. We heavily considered this mechanism, but requiring two motors to spin each of the brushes was a negative in our eyes, as we were trying to minimize the size and weight of the pickup mechanism. Finally, we settled on an approach that uses a single motor to spin a blade which guides the ball up a ramp and into the ball container. In terms of simplicity to build, effectiveness, and size/weight, this method seemed the most optimal.

Computer Vision

Over the course of the project, our team iterated through a number of techniques for isolating tennis balls in a low-quality image. We ended up using a combination of Canny edge detection and thresholding, but we tried many other solutions as well. Contour detection on each channel of the image, as well as on the grayscale version of the image itself, found too many contours and rarely identified the ball itself. Hough circle transformations had the same issue, as did the OpenCV blob detector. Thresholding alone worked well enough in a controlled test environment, but changes in lighting conditions, similar colors in the background, and random noise would result in wild swings in accuracy. Likewise, Canny edge detection would consistently draw an edge near the tennis ball, even against low-contrast backgrounds, but it would also find edges everywhere else. In addition, the tennis ball edge would often be incomplete or segmented, and so we were unable to use the length and area of the contour to check for circularity. We tested image smoothing as a way to combat noise, trying Gaussian, box, median, and bilateral blurring with varying kernel sizes. We settled on a median blur with a kernel of (5, 5) for a balance between performance and noise removal. We also looked into mean-shift segmentation for image preprocessing, but the technique was too expensive for us to utilize, as we were unable to maintain a framerate of even 1 fps at low resolutions. The decision to pair thresholding and Canny edge detection together was due to these two techniques detecting different sets of false positives. We found that examining the overlaps in the results returned from each method often resulted in only tennis balls being found.

System Description

Camera and Computer Vision

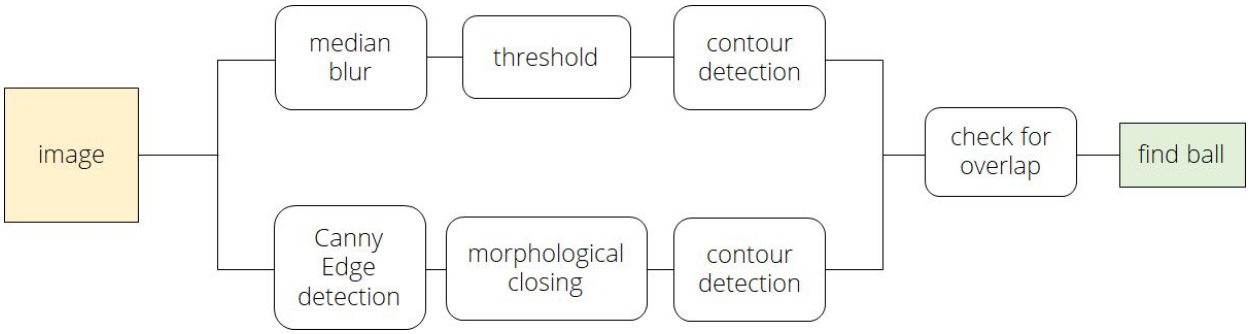

Figure B - a block diagram of our ball detection algorithm

Our bot constantly scans the court to determine where and how to move. To do this, we use a SainSmart Pi Camera with a wide-angle lens to provide a 160-degree field of view. The camera has a ribbon cable which directly attaches into the CSI port of the Raspberry Pi. We access a live video stream from the camera using the Python picamera library. The video stream has a resolution of 480x368 and runs at 4 frames per second, which provides a good balance of information and responsiveness.

For detection, we use OpenCV, an open-source computer vision library. In each frame of our camera's video stream, our Pi isolates tennis balls and localization markers from the environment. Our ball detection strategy consists of two parts: thresholding and Canny edge detection. Thresholding is a simple way to efficiently eliminate most of the image and find objects that lie in a certain yellow-green color range. As there are not many objects on a tennis court that are similar to the yellow-green of a tennis ball, this method will be effective in isolating our targets. However, we also incorporated Canny edge detection to ensure that the objects picked out from color thresholding are not walls or other false positives. We smoothen our image before thresholding by using a median blur with a 5x5 pixel kernel. This reduces the amount of noise in the image without losing a significant amount of color detail. We also dilate and then erode the output of Canny edge detection to group clusters of edges together. Canny edge detection rarely finds a complete circle outline of a tennis ball, so this morphological closing operation helps make the outline more complete while also reducing the number of contours processed in the next step. After finding yellow-green pixels and possible edges in the image, we compare contours found in each set of results and only output contours that have matching centers. Our localization marker strategy is similar, but we also include a shape

invariant to verify that the bounding box of the contours is twice as tall as it is wide to match the unique shape of our markers.

Bot behavior

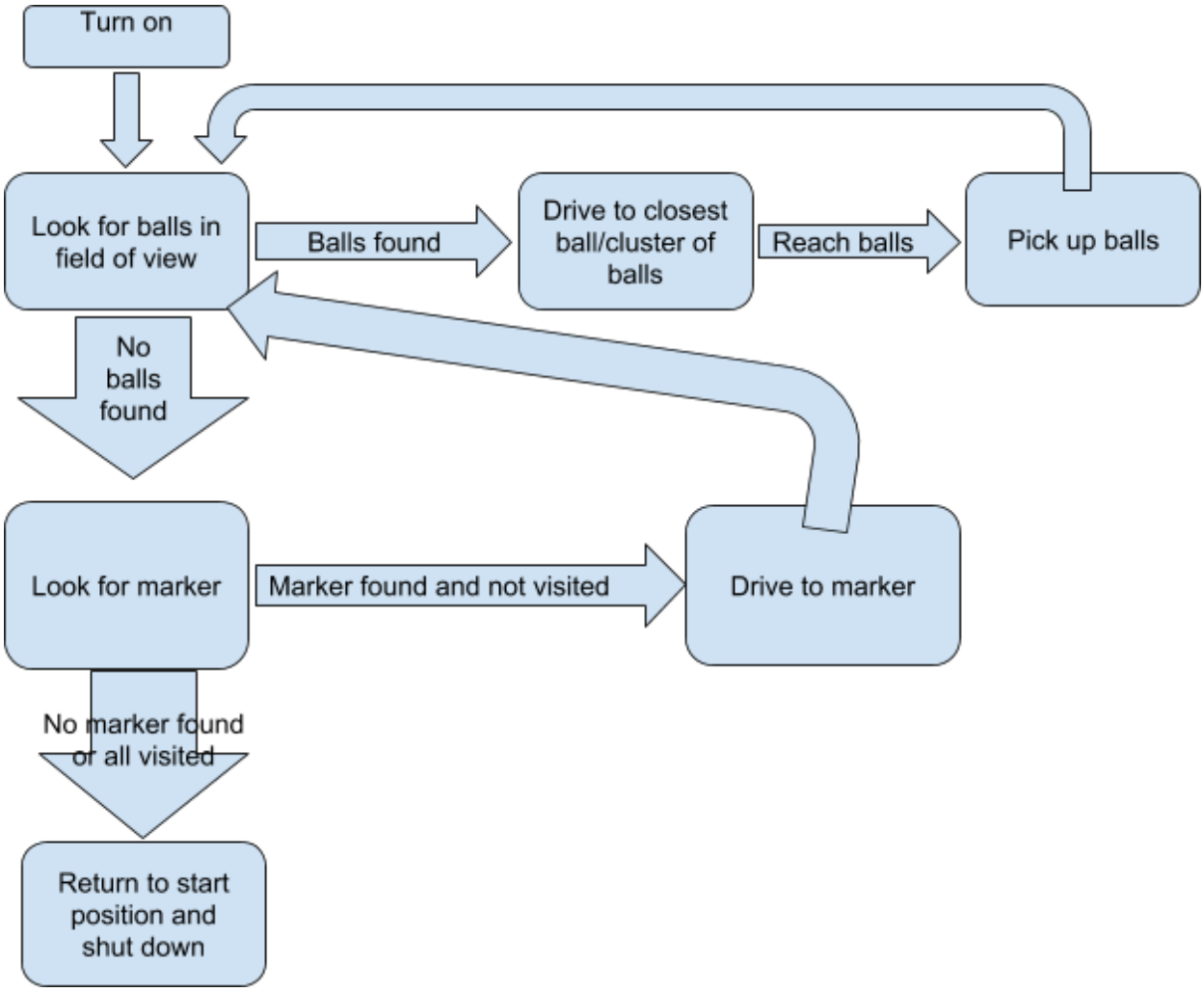

Figure C - a block diagram for the AI

While the player is playing, the robot will be turned off. Whenever the player wants to take a break, or runs out of balls, they can turn on the robot. Once turned on, the robot is controlled by a finite state machine. It starts off looking for tennis balls within its field of view. It observes its current field of view for 10 frames to account for errors in the ball detection and inconsistencies in lighting. If it doesn't see any balls in this time, it rotates 90 degrees to the right and continues

looking for balls. While it can't find balls, the car will keep rotating by 90 degrees until it reaches its original position. It then looks for markers, which are placed at the 4 corners of the court. These markers are built so that the bot will be able to find at least one of them from all possible positions. The bot will go to any marker that it hasn't visited in its current run, and repeat the process to look for balls again. If it has visited all 4 markers and not found a ball, the robot assumes that there are no balls on the court and the run ends.

If the robot finds one or more balls, it will start to move towards the closest ball. When the ball is less than 8 inches away, the robot will engage the pickup motor and continue driving towards the ball. The ball is assumed to be picked up once it is no longer visible for 3 frames. It then resumes looking for balls, before which the FSM makes sure the pickup wedge is pointing downwards so that it doesn't obstruct the robot's field of view.

Localization

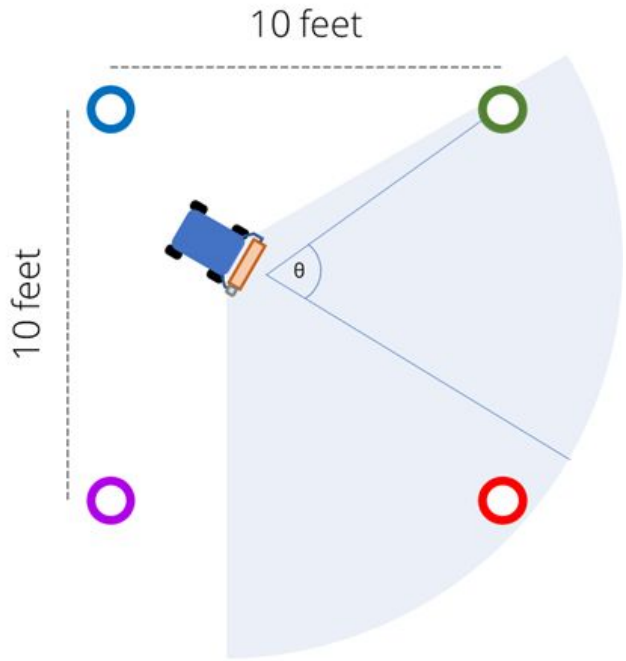

Figure D - The demo setup

In addition to running the FSM, the robot also localizes every frame. This is possible because at least one marker is visible at all points of the court and their positions are known. Another key feature of these markers is that they are all different colors, which means that the robot will be able to determine the marker's position if it detects it. The robot is always looking for all 4

markers. Once a marker is detected, we estimate the distance between the centroid of the marker and the robot, and also the angle θ between them. If only one marker is detected at a certain frame, the robot's position is simply the position calculated based on that marker. However, if multiple markers are detected, the robot has multiple positions that it has calculated. We use the previously determined position to assign a confidence to the multiple positions calculated since the robot was estimated to move roughly 6 inches per frame. This confidence is used to prevent error due to false detections of markers.

To represent where the robot thinks it is, we also wrote a very simple GUI that would represent the position of the bot on the court to the closest foot. This is because the error bound on our localization is approximately 1.1 feet. This GUI reads from a positions text file that is being written to by the FSM at every frame. It renders the robot's current position as a '#' character while all the positions in the grid are represented by the '.' character.

PWM Control

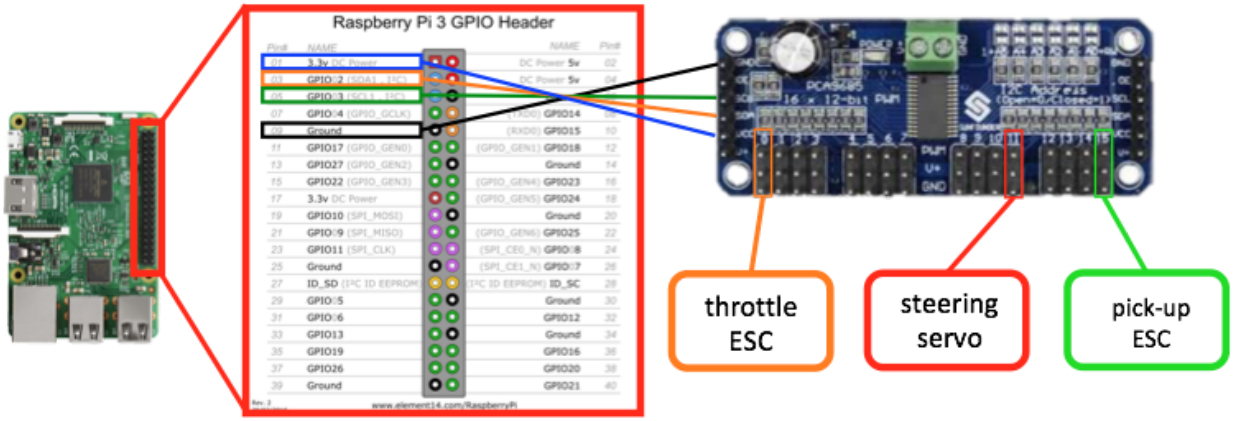

Figure E - the wiring setup between RPi, servo shield, and electric speed controls/servos

Figure E shows the configuration of how the PWM signal travels and controls the different motorized components of our system. We are using the 3.3V pin and ground pin from the GPIO pins of the Pi to power our PCA9685 16-Channel PWM servo shield. Pins 3 and 5 on the Pi are also connected to the servo shield as those are the I2C pins for SDA and SCL. The SDA and SCL pins are sending the PWM data and clock that we send to the peripheral motors. The servo shield supports up to 16 channels but we only use three. Channel 0 of the servo shield connects to the throttle electric speed control, which controls the car's forward and back movement. Channel 15 goes to the steering servo, which controls how the wheels turn. Channel 11 is used for the pick-up mechanism electric speed control. This controls the gear-motor which

runs the ball pickup mechanism. Each channel on the servo shield contains three pins. In order from top to bottom, they are: data, power, and ground.

For the throttle ESC, our PWM signal controls both how fast and in which direction (forward/back) that the car moves. The PWM signal for the steering servo turns the front wheels either left or right. For both the ESC and steering servo, the PWM signal has a pulse width of 1ms to 2ms, which corresponds to a duty cycle that ranges between 5% and 10%. In other words, the signal is high for only 1ms to 2ms and cycles at 50Hz. A duty cycle of 5% for the ESC results in the car reversing at max throttle. Likewise, a duty cycle of 10% results in the car accelerating forward at max throttle. For the steering servo, a duty cycle of 5% turns the front wheels left, and a duty cycle of 10% turns said wheels right. A duty cycle of 7.5% is neutral for both steering servo and ESC. For the pick-up electric speed control, the PWM signal simply controls the speed of the gear-motor from 0% to 100%. We determined 75% power to be sufficient in smoothly collecting the ball, allowing for some padding to avoid overdriving the motor.

Ball pickup mechanism

Figure F - (left) a side view of our design. (right) a top-down view of our design.

The final subsystem for our bot is the ball pickup mechanism. This is a motorized system using the ball pickup gear-motor mentioned earlier. Figure F shows the strategy we use to corral the balls. A gear-motor rotates a cylinder above tennis ball level that has a blade attached in order to contact the ball. As the car moves forward and detects that the ball is getting close to the front of the car, the gear-motor, controlled by an electric speed controller, rotates the cylinder. The blade then hits the tennis ball, moving it up a curved ramp and pushing it into the ball holder, which is mounted on top of the car and can hold up to four balls. As mentioned earlier, the ball pickup mechanism does not constantly run but rather turns on when the computer vision determines that the ball is nearing the mechanism. This is so that we save power and also

minimize the effect that the pickup gear-motor rotation has on the car's movements. In addition, our system, again using computer vision, makes sure the mechanism never comes to a stop with the blade sticking up, as this situation blocks the camera view. Regarding the gear-motor, we decided to go with a high powered gear-motor, which is a motor that has a metal spur gearbox attached to it. The reason we use this instead of a normal DC motor is that a gear-motor converts RPM into torque, allowing us for a much higher torque provided by the motor. For the mechanism, we do not need a high RPM at all but do need a decent amount of torque to provide enough force to push up the ball, so a gear-motor is perfect for our situation.

Evaluation

We measure the performance of our robot at its various tasks using the following metrics:

Task: Ball detection

All the below metrics are measured with a video frame rate of 4 fps.

Maximum detection range: In ideal lighting conditions, the robot can detect a ball that's 5 feet away. The higher the range, the better the robot's performance since it can detect balls that are further away and will thus miss fewer of them.

Detection accuracy: This is an important metric since we want to minimize false positives and failed detections. Our robot does quite well in terms of minimizing false detections, but has a slightly higher false positive rate. The numbers are shown in the below confusion matrix.

# of experiments = 74	Detected: No	Detected: Yes
Actual: No	25	11
Actual: Yes	2	36

Task: Ball pickup and transport

Pickup speed: This metric was tested by putting balls in a straight line and have the robot pick them up. Our robot can pick up balls at a speed of 1 per second.

Pickup success rate: When the robot attempts to pick up a ball, it succeeds 75% of the time. This metric is important since we want to maximize the success rate to ensure that the robot doesn't constantly drop balls.

Maximum transport capacity: Even though our pick up mechanism is designed to carry a maximum of 4 balls, the robot can generally carry only a maximum of 3 due to the balls moving around in the box. However, it is able to successfully carry 4 balls 10% of the time.

Task: Robot movement

Average speed: Our robot goes at an average speed of 5 mph, which is faster than the average human walking speed (4 mph).

Rotation time: Our robot takes 12 seconds to turn 360 degrees.

Driving accuracy: When directed to drive to a specific location, the robot is able to successfully reach this position 80% of the time.

Task: Localization

Estimation accuracy: While testing our robot, we found that our estimated positions were off by a maximum of 1.25 feet when the marker detected was 8 feet away. As the robot got closer to the marker, this error decreased.

Project Management

Schedule

	2/25	3/4	3/11	3/18	3/25	4/1	4/8	4/15	4/22	4/29
Rohit	Ball detection/computer vision techniques			Car behavior and FSM		Localization design/development		Computer FSM testing		Presentation /demo
Ben	Car control/ movement API		Car movement calibration/ improve computer vision			Pick-up control	Design/build car mounts	Car vision testing		Presentation /demo

David	Car control	in-place turn	Design/build pick-up mechanism	Connect pick-up mech to system	Ball pickup testing	Presentation /demo

This is was the breakdown of how our project schedule turned out. At the start of the project, one of our focuses was gaining full control of the RC car as that is the basis of our project and most of our other parts would need to utilize the car movement. In addition, we made sure to begin our ball detection/computer vision development early. This proved to be a smart decision as there was a lot that needed to be worked on and improved throughout the project in terms of computer vision. There were many different techniques, talked about earlier, that were attempted and tested. As the movement and vision of the car progressed, we began working on the pick-up mechanism and localization for the car. These were the main components of our system, and we used the final couple weeks to finalize the construction of the self-contained system as well as running many tests and adding improvements from how the tests ran.

Team Responsibilities

Our project utilized software, embedded systems, and hardware. We split our responsibilities so that we each take on one of the components to be the the primary role to tackle. Rohit focused on the software which included the high-level FSM that controlled the behavior of the car as well as computer vision for ball detection and localization. Ben took charge in the embedded parts, gaining full control of the car's movement through the Raspberry Pi. David worked primarily on the more hardware and mechanical components such as ball pick-up mechanism. However, for almost all the tasks, it ended up that we always had at least couple people working together on it, as we found our progress to be much more successful by having multiple brains trying to solve the problem. For example, Rohit and Ben worked hard together on trying to improve our car's vision, figuring out the effectiveness different techniques and combination of techniques. This team effort allowed each member to be able to get experience in all areas of the project instead of just solely focusing on their assigned parts.

Budget

Item	Description	Cost
1/10 RC Car	Vehicle to move the system and collect balls; all parts are attached to the car chassis	$119.99

Raspberry Pi 3	Controls the system; runs OpenCV; determines pathing and movement; uses PWM to control car movement	$39.99
SainSmart Pi Camera	Wide-angle camera compatible with Raspberry Pi; feeds live video into Pi for ball detection	$26.99
PCA9685 Servo Shield	Interfaces between the Pi and both the RC car (steering/throttle) and pickup mechanism	$11.99
USB battery	Powers Raspberry Pi	$25.49
LiPo battery	Powers pickup mechanism motor and ESC	$26.99
Electric speed control	Controls signals sent to the pickup motor	$20.99
370 brushed motor	Originally used as our pickup motor, blew out	$9.56
75:1 high power gear-motor	Now used as our pickup motor, has lots of torque	$36.95
Miscellaneous parts	Replacements, small additions, chargers, cables, 3D prints	$199.68
Total		**$81.38**

From our $600 dollar budget we ended our project with **$81.38**. A decent chunk of the budget was used towards the latter half of our project to purchase backups and spares in case something went wrong during our tests or demo.

Risk Management

Hardware level latency:

This is an issue which we predicted to run into from the start of the project. As we used several different pieces of hardware (Raspberry Pi, servo shield, RC car) and have them connected to each other, there is always the risk that there may be some latency in the communication between the components. And since we planned to have the video feed control the car in real-time, hardware latency was an anticipated issue.

Although we did see some signs of latency, we luckily did not run into latency problems that significantly affected our systems. We made sure to build an efficiently connected system from our code to our wire connections.

Insufficient processing power:

This risk dealt with our Raspberry Pi and its ability to run our code. As the Raspberry Pi is not a very strong computer, we ran into the issues of there not being enough processing power to handle all our processes. One of our methods to help was dropping the frames per second of the video feed close to the minimum value that still allowed the system to work as intended. In addition, we tried to write efficient code to lessen the processing load, but we did end up running into problems with this relating to our computer vision. Many of the computer vision and detection algorithms we tested were computationally complex and required significant processing power. We did a lot of testing and development to develop the most effective computer vision technique we could within the bounds of our Raspberry Pi.

Hardware failures:

Like hardware latency, this was another risk we saw relating to the hardware components of our system. Whether it is due to poor manufacturing or something going wrong during our project, there was always a chance during our project development that some parts could break or fail. We made sure to maintain our parts well and also handled them very carefully, making sure nothing shorted or broke. Our tests were run in safe environments with padding or with someone always ready to restrain the system in case it somehow ran errant. In addition, we used some of our budget to order spares for parts we deemed most volatile so in case something did break, we had a back up. A situation where a hardware failure set us back a bit was in the case of the pick-up mechanism motor. At first, we used a normal DC motor to try to spin the pick-up mechanism. However, after we set up the mechanism and tried to have it pick-up the ball, the motor immediately smoked and burned out. We realized that we had an oversight with the mechanical properties of the motor, as the normal DC motor provided very little torque, something which was necessary to move the ball up. After careful researching, we purchased a gear-motor (and a back-up for it) that ran our pick-up mechanism successfully as we talked about in an earlier section.

Environmental conditions:

Environmental conditions was a big risk for us coming into the project because we knew a significant component was our car's vision and ability to find objects of interest. Lighting was the big unknown we had as natural light constantly changes, which affects our testing environments. We worked on managing that risk by purposely testing our system and vision different lighting environments such as natural daylight, bright indoor lights, and dimmer indoor lights. We were not able to make our car's vision dynamic but we were able to set up our system so that we could easily adjust some settings to account for the current lighting environment.

Related Works

Tennibot

The Tennibot is a commercial implementation of a robot that picks up tennis balls. It is slated for a 2018 release, and has a corresponding smartphone app and an ergonomic design for carrying. It can hold up to 70 balls, but its main weaknesses are price and speed. It moves at a 1.4mph pace, and it costs $899 per unit.

Ballbot

An experimental robot created over at UC Berkeley, Ballbot was designed as a replacement for the human ballboy. Like its fleshy counterpart, Ballbot detects tennis balls in movement and adjusts accordingly. Ballbot is supposed to be able to perform localization, as well as avoid human players and determine when a point is over. It is built on top of a 1/10 RC car and looks very much like a prototype. Its main flaw is that it assumes that there is only ever one ball on the court at a time, and it can only store 3 balls. Also, there are very few videos of it functioning as promised.

TBRS

We intended to tackle some of the weaknesses of the previous two designs with our robot. TBRS moves quicker than a Tennibot while also having more carrying capacity than a Ballbot. In addition, TBRS is cheaper than both rival platforms. Most importantly, TBRS gave our team valuable experience in combining computer vision and embedded systems into a functional product, which is something that could not have been achieved by buying one of the other devices.

References

"About Donkey." Donkey Car, docs.donkeycar.com/.

"Camera Module V2." *Raspberry Pi*, www.raspberrypi.org/products/camera-module-v2/.

"ESC and PWM Signal Pulse Width." RC Groups, 1 June 2010, www.rcgroups.com/forums/showthread.php?1253674-ESC-and-PWM-signal-pulse-width.

"OpenCV Tennis Balls Recognizing Tutorial." Elphel, 27 July 2010, www.elphel.com/wiki/OpenCV_Tennis_balls_recognizing_tutorial.

Rosebrock, Adrian. "Accessing the Raspberry Pi Camera with OpenCV and Python." PyImageSearch, 14 June 2015, www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/.

"Smoothing Images." OpenCV: Image Thresholding, docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html.