ARioKart

Authors: Sourav Panda, Ranganath Selagamsetty, David Yang

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—An augmented reality racing game with networked remote controlled vehicles. The basic concept of the game is similar to Mario Kart, with a slalom-style user-specified race track and various virtual items and power-ups to augment the physical car race. The system consists of cars, gates, and computers that run the game software and render the AR display and interface. Compared to similar existing AR games, this game would provide a more engaging AR experience with meaningful interactions with the environment.**

***Index Terms*—Augmented Reality, Computer Game, Computer Vision, Infrared Detection, Object Detection, Remote Control**

I. Introduction

A new frontier in human-computer interaction, augmented reality is a technology that uses an interactive display of the physical environment to enhance the user's real-world experience[1]. Augmented reality has the potential to transform everyday scenery into an information rich environment for new immersive gameplay experiences. Modern augmented reality games fail to reach the potential of the medium as many of them confine the user experience to static interactions with the background. This often amounts to traditional games simply being overlaid onto a video stream without taking advantage of the visible features of the current environment.

The solution in this paper, ARioKart, aims to improve upon existing AR games by unifying physical components that interact with the surroundings and game software that augments the display. ARioKart is a slalom-style racing game with physical cars and gates and virtual items. Users interact with game software on their laptops to operate remote controlled cars and see an augmented video stream from the cars' perspective. Though related products, such as drones and ordinary RC cars, exist, they have many shortcomings, including their cost and battery life, and lack the AR aspect. ARioKart provides all the fun of traditional RC cars with the added immersion of the first-person Augmented Reality component.

II. Design Requirements

In order to deliver a high-quality AR racing game experience, the implementation should meet the following software and hardware design requirements.

A. Software Requirements

Latency

The primary design requirement for the software component of the project is to ensure a seamless user experience. As such, the video streaming component should deliver a clear picture with at least 30 fps and a latency of no more than 100 ms, and controller input should be processed and acted upon by the cars with a latency of less than 100 ms as well. This value was derived from the related field of online gaming, where 100 milliseconds is considered the upper bound for acceptable latency in gameplay. A delay greater than this could induce nausea as well as frustration due to lack of responsiveness. This will be tested by measuring round-trip time within the user and the car, and extrapolating one-way latency from this.

B. Hardware Requirements

Battery Life

To meet the high-level design goal of having a higher battery life than commercial drones, which can fly for around 10-20 minutes on a charge, the cars should be able to operate for at least 30 minutes continuously.

Top Speed

The top speed of the cars should be at least 3 mph to ensure that ARioKart can deliver a fast-paced first-person driving experience without being too fast to control or follow on foot.

Motor Speed Control

Each car must must have a controller that can set the motor speeds with enough accuracy to drive the car straight for 20 feet, the size of the demo space, with less than half a car width of deviance from the center line. This allows for standardization in speed control and ensures that no car has an unfair advantage due to hardware discrepancies.

Turn Radius

The turn radius of the cars must be less than five feet at their base speed so that the cars are able to make at least two 180 degree turns within the 20-foot demo area.

RFID Detection Speed

Each car will have an Radio Frequency Identification scanner mounted to the bottom of the car that will be able to detect a gate before the car completely passes over it at base speed, i.e., a latency of no more than 121 ms (derived given the size and the top speed of the car as well as the location of the scanner on the car, 6.4 in before the rear bumper).

$$latency = \frac{6.4\ in\ to\ rear}{3\ mph * 63360\ in/mi\ /\ 3600\ s/hr} = 121\ ms$$

This allows the game software to accurately determine which place each car is in after passing through a gate.

IR Range

Each car will have an infrared receiver that should be able to detect an infrared blaster mounted on another car across the demo area at a distance of 20 feet. This will allow cars to detect opponents ahead for using items.

III. System Architecture and Interactions

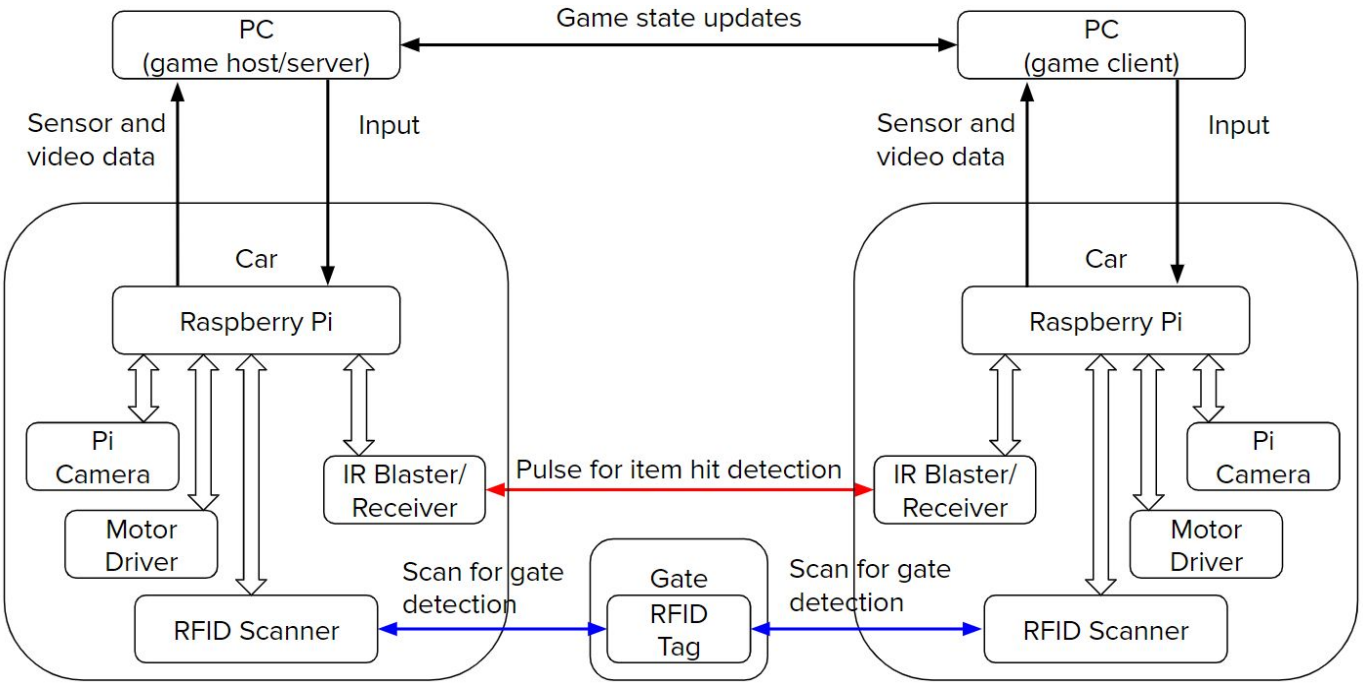


Fig. 1. Overall system architecture block diagram

A. Overall Architecture

The general design of the system follows the architecture depicted in Fig. 1. From the highest level to the lowest, the physical components are PCs that run the game software and display the AR overlay, Cars with Raspberry Pis that control the motor, IR, RFID, and camera modules, and gates that are scanned by the cars. Each PC is connected to one car and the game host. There are four major communication pathways between the physical components in the system as shown in Fig. 1, with communications over Wifi in black, IR in red, and RFID in blue:

1. *Between two PCs*: Each PC runs an instance of the game, with one acting as the game's host and the others as clients connected to the host. Communication between PCs consists of game state updates with item use and gate information from their respective cars.
2. *Between a PC and a car*: Communication from the PC to the car consists of player input with commands for the physical components on the cars, such as commands to set motor speed or commands to fire the IR blaster. The car sends to the PC a constant stream of video and discrete sensor data from the RFID and IR modules.
3. *Between two cars*: A car following another car in a race can fire its IR blaster at the IR receiver on the car in front, causing it to stop if a valid item was used.
4. *Between a car and a gate*: Cars passing over gates will scan the gates to determine their position in the race and on the course.

Detailed communication diagrams for the two major interactions of item usage and gate scanning follow.

B. Item Interaction

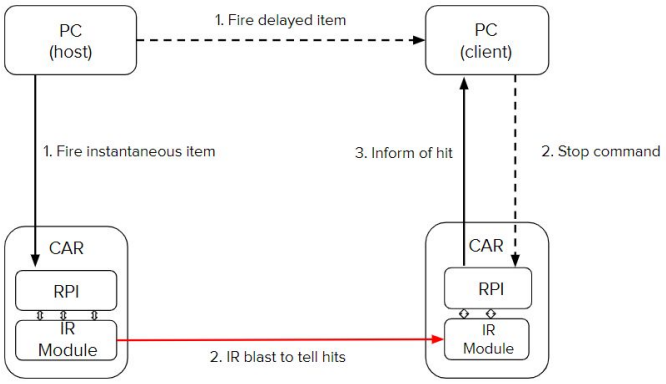


Fig. 2. System interactions for item usage

The flow of interactions depicted above follows two different paths depending generally on the type of item used: instantaneous items and delayed items. Instantaneous items, when fired by the user, take immediate action, sending a command down to the car to fire a IR blast. This blast, if a hit, will be registered by other Car. A delayed item flows in the opposite direction. Some items require greater knowledge of the game state, for instance place, and therefore must be passed through the PC Host. These are then forwarded to the target. The target PC then forwards the correct consequence to

the car.

C. Gate Interaction

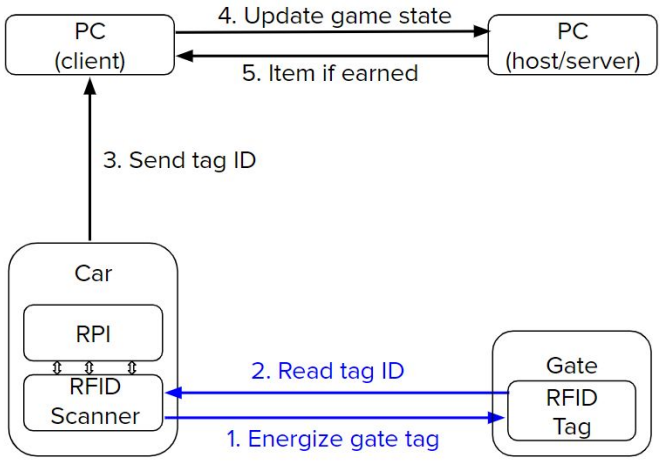


Fig. 3. System interactions for detecting passed gates

When the RFID Scanner in the car first passes over the gate, the RFID Scanner will energize the tag and read the bit pattern of the tag ID in the gate. This gate ID is then wired to the onboard computer. The onboard computer pass this ID to the the PC that it is networked with. The PC relays this information, as well as as a timestamp of when the ID was received to the other PCs running game instances to be able determine what place the car is in. Since game-item distribution happens only when a car passes over a gate, the "host" PCs will provide the "client" PC an item, if that user has earned a item.

IV. Design Trade Studies

In order to deliver a high-quality AR racing game experience, the implementation should meet the following software and hardware design requirements.

A. Software Requirements

Latency

For the latency requirements the primary target was 100 ms end-to-end latency on the video stream with a clear picture and infrequent lag spikes. Factors that influenced the latency included video encoding time on the Raspberry Pi, network transit latency, video decoding time on the PC, and rendering time in Unity. Of these, the aspects we had control over were the network latency, and rendering. We tested the latency by displaying and recording a timestamp on the PC screen and checking the time difference between the two while varying the quality of the video stream and the rendering technique. Our results are recorded in Table I.

TABLE I. Latency Test Results

Test Parameters			Latency Result
Resolution (16:9 aspect ratio)	***Frame rate***	***Rendering Engine***	
1080p	30 fps	Vuforia	600 ms
720p	40 fps	Vuforia	400 ms
720p	40 fps	Unity	330 ms
576p	40 fps	Vuforia	270 ms
576p	40 fps	Unity	200 ms

Our final result was a latency of 200 ms with a resolution of 576p at 40 fps using Unity's video rendering. We believe this is close to the best video streaming latency we could achieve with OpenCV as a test running a minimal OpenCV demo application had a 170 ms latency with the same video quality. We originally considered using a different library for video decoding, but decided that it would be too difficult to implement and unlikely that we would be able to achieve better results than OpenCV.

Another latency requirement we had was that control and sensor data be transmitted with a latency of less than 100 ms. We measured this by recording the time for Xbox controller inputs to be sent to the Pi and acknowledged and found that our initial implementation with TCP sockets achieved a round-trip latency of 16.5 ms, giving us a one-way latency of ~8ms, well within our targeted range of less than 100.

B. Hardware Requirements

Battery Life

Our requirement for the batteries for our cars were that should allow the car to operate for at least 30 minutes. The battery life of our cars depended on the following factors: max current draw from the motors, max current draw from the raspberry pi. Based on the specifications of the raspberry pi we ordered, and the motors we chose, we expected the max current draw out of the battery pack to be 5.25 Amps (1.25 Amps from the raspberry pi, and 2 Amps from each motor at full power). To meet the our design goal of having of at least 30 minutes, we selected a 6000mAh battery pack that had 5V/12V dual supply ports. We tested the battery life by running the motors at full power (100% duty cycle) and running the pi with a full software workload (video server, TCP connection to PC, continuous read from encoders) and having a script on the pi write a timestamp to a log file. Upon inspection of the log file after rebooting the pi (since the pi shutdown once the battery ran out of power) we saw that battery was able to provide 7 hours and 14 minutes of steady power. While this is a significant improvement of what our goal was, the test was conducted while the motors were free spinning, and does not account for the motors drawing more current if they running on the ground against friction. We realize that this was not the most accurate way to observe the battery life of the car, but we didn't have an appropriate

testing setup that allowed us to control the car while it drove under load.

Top Speed

Our requirement for the top speed was that the cars should be able to reach a top speed of no less than 3 mph. To be able to meet this requirement, we purchased motors that were advertised to be able to drive up to 600 rpm (potential top speed of 4.5 mph). We tested these motor by reading the maximum value from the encoders when driving a 100% duty cycle to the motor. We saw that the motors were able to drive at a maximum of 500 rpm. With 2.56in diameter wheels, this gave our cars a top speed of 3.81 mph, thus meeting our initial requirement.

Motor Speed Control

Our requirement for motor speed control was that the cars deviate from the center line by less than one car width per 20 feet of driving straight. We attempted to mitigate deviance errors by designing a 3D model of the car and using a precise laser cutter to make the frame of the car and by tuning the PID controller that controlled the motors. We tested this requirement by driving the cars along a 20ft straight path, and measuring how far the car deviated from the center-line it started driving from. We saw that on average, maximum deviation that cars drifted from the center-line was 5 in, which is about ⅔ the width of a car, and thus meeting our requirement.

Turn Radius

A trade-off was made between a more accurate Ackermann steering system and a Differential System. For our project we chose to go with a differential steering system, as the Ackermann steering system added mechanical complexity to an already mechanically complex project. Ackermann steering kits were also much more expensive than their differential counterparts. This choice, however, came at the cost of accuracy in turning. We believe most of the issues with turning came from a lack of traction, so we attempted to increase traction by driving on carpeted surfaces. The end result was that at full speed the car was able to turn with a radius of about 5.5 feet, a marked improvement from the greater than 10 foot turn radius on tiled floors. However, we were unable to make the car turn at anything other than very gentle angles, or in place, rotating around a point between the front-wheel motors.

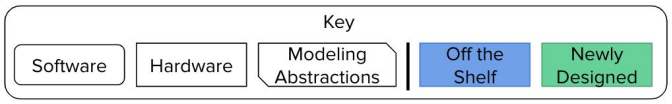
RFID Detection Speed

The requirement for the RFID readers was that they were able to detect a gate before the car completely passed over it at the base speed of 3 mph. The biggest factors for being able to read an RFID tag at speed are how much power the reader can source and the sizes of the reader and tags. Originally, we tried a Raspberry Pi compatible HiLetgo RFID readers and NFC stickers. Unfortunately, this pair was unable to meet our requirements. We then tried larger area readers and credit-card bit tags. We saw an improvement in performance, but still were unable to meet our requirement. The reader we currently have in our cars simply does not have enough power to be able to energize the tags and read the bit pattern in under 120 ms (our initial requirement). We tested this by driving the cars over the gates multiple times at different rpm settings. Once we were able to drive over a gate and detect the gate 10/10 times, that speed setting was noted to be reliable. We then proceeded to increase the speed setting of the car to see what the maximum speed we could drive the cars, yet still be able to detect a gate underneath 10/10 times. We found that the fastest we could drive the cars over the gates while still being able to reliably detect the gates was at 100 rpm (0.76 mph). Faster RFID readers similar to those used in RFID race timing systems were too expensive for our project, and we unfortunately did not have enough time to experiment with making our own tags to improve performance.

IR Range

Our requirement for the IR subsystem was that a receiver should be able to detect a pulse from another car's blaster from 20 feet away. The range of the IR is dependent on the on the amount of current pumped through the IR LEDs. The original circuit design allowed up to 0.5 Amp spikes to flow through the IR LEDs. However, we found that at this current and 1.38 kHz pulse setting, the drain resistor (R5 in Fig 7) was unable to operate correctly (resistor meltdown). To compensate, we increased the resistance of the drain resistor, and dropped the current to spike at 150 mA when on. This allowed the IR pulse to draw less power, but reduced the distance the pulse could travel. If we had more time and funding, we would look into purchasing a low-resistance ceramic resistor to allow more power to be safely delivered to the IR LEDs. We tested this by continuously firing the blaster from one car and varying the distance of the car receiving the direct blast. Once the maximum centerline distance that the receiver car could reliably detect the blaster, the car was moved horizontally to determine the cone of effect of the IR pulse from the blaster. This test was conducted at night, when there was as little interference from sunlight as possible. It was found that the IR pulse was able to be detected up to a centerline distance of 7ft, with a cone of effect of 4.5in. Given more time, we would've liked to explore the possibility of reducing the number of IR LEDs in parallel. The IR LEDs we purchased where rated to allow up to 1 Amp spikes. If we were able to procure a ceramic resistor, a circuit could be designed to allow 1 Amp spikes through this resistor, while delivering more current to the fewer IR LEDs. While there would be fewer than seven LEDs, the range distance might improve due to the increase in power delivered to each individual IR LED.

V. System Description



A. PC Software

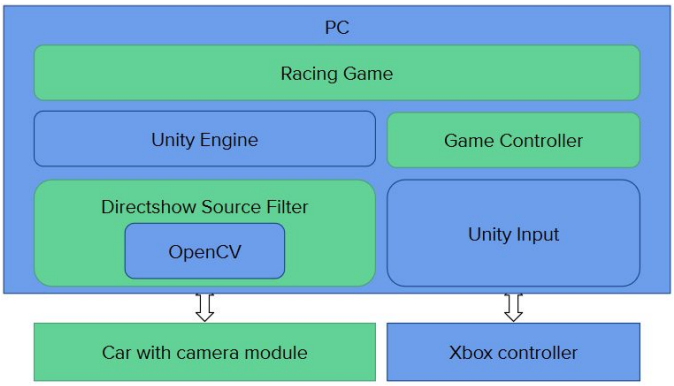


Fig. 4. Subsystem diagram for PC

The software run on the PCs can be separated into three modules: game, video, and controller.

1. *Game*: The game software manages state and operations related to the race including information about items and player place and progress. It is also responsible for displaying a car's video stream and rendering an AR overlay with race and item information. The game is implemented in C# using the Unity game engine and Unity Multiplayer for multiplayer communication between game instances.
2. *Video*: The video module is responsible for receiving the video stream from the Raspberry Pi on the car and rendering it to the screen in a way compatible with the AR engine. For this implementation, this module takes the form of a C++ Directshow source filter, which uses the Directshow Windows API to act as a virtual camera device recognizable by Unity. The source filter receives raw H.264 compressed video from the Pi through TCP/IP over Wifi, decodes it with the OpenCV library, then draws it to the screen.
3. *Controller*: The controller software receives input from the player, converts it to commands for the car, and sends those commands through TCP sockets to the car, which then controls its physical components. Players interact with their PCs through Xbox controllers, which interface with the controller software using Unity's Input module. The controller module also receives RFID and IR sensor data from the Pis, which leads to game state updates in the game software.

B. Onboard Software

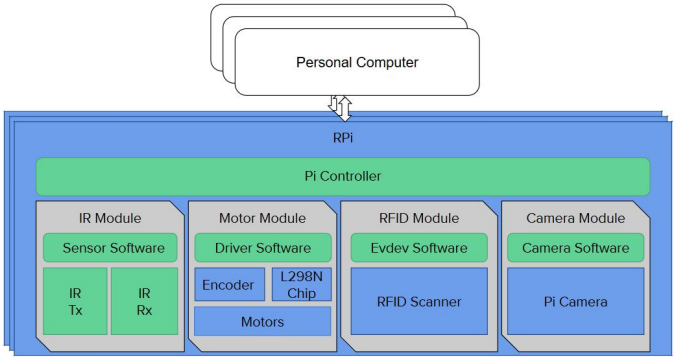


Fig. 5. Subsystem diagram for Onboard Controller

The software run by the onboard computer and hardware components on the cars can be separated into five discrete modules: Controller, IR, Motor, RFID, and Camera.

1. *Controller Module*: Each Raspberry Pi has software that receives data from the four other modules in the car and transmits this data over a TCP connection on Wifi to the PC that the onboard computer is networked with. The controller software receives inputs to control the movement and actions of the car and transmits whether the IR receiver was triggered, the gate ID when passing over a gate, the current speed from the motor encoders, and the video stream from the camera.
2. *IR Module*: The Raspberry Pi has IR sensor software that interfaces with the IR circuits on the car. This software will receive commands passed from the PC through the controller software instructing the car to fire the IR blaster. The IR sensor software will drive voltage levels on the GPIO pins to pulse the IR blaster. This software will also alert the controller software if the IR receiver circuit has been triggered by an opponent's IR blast.
3. *Motor Module*: The Raspberry Pi will have software that interfaces with the motor's encoder chip as well as the motor driver chip. It translates commands from the PC into a PWM signal for the L298N motor driver circuits. The controller software, motor driver circuits and Hall-effect encoders create a feedback control loop that allows the onboard computer to accurately set speed through PID control.
4. *RFID Module*: The Raspberry Pi has software that interfaces with the RFID reader on the car. The controller software interacts with the circuit via the Linux input event device interface as the RFID reader emulates a keyboard. The RFID reader continuously reads and only alerts the controller software when the scanner has successfully detected a NFC tag ID from a

gate.

5. *Camera Module*: The Raspberry Pi has software that interfaces with a Pi Camera V2. The camera supports live-streaming video in h.264 video encoding format. The module is nothing more than a pipe for this video to the controller to relay to the PC where it will be processed.

To allow the simultaneous reading, receiving, and transmitting, of data, the programs for the various modules were run on four separate threads with data and commands shared between them using mutexes (Fig. 6.). The control thread runs part of the controller module, receiving commands from the game and controlling the motor speed setting and IR blaster. The motor thread and RFID threads run their respective modules and make their data available to the data stream thread, which sends sensor data to the PC over the network and also reads the IR receivers.

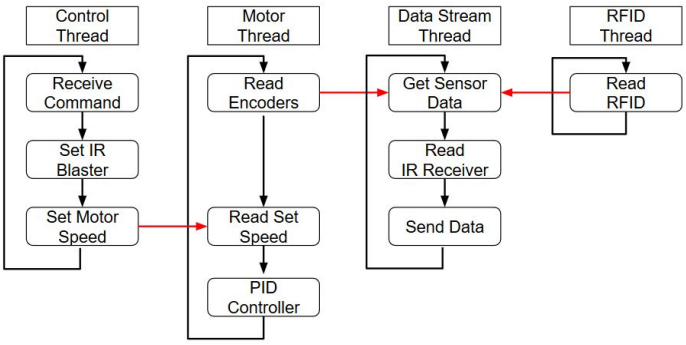


Fig. 6. Thread organization on the cars

C. *Hardware and Manufacturing*

1. *Mechanical Design*: The physical layout of the car was designed to take into account the various mechanical constraints of the modules described above. These constraints included placing the RFID scanner close to the bottom on the car to allow for more accurate gate reading, evenly placing IR receiver circuits around the car to allow item usage on three sides of the car, distributing weight evenly from the center of mass to prevent drift during normal use, and placing the motor driver circuit in a location that has easy access to air-cooling for better motor control performance. Seven IR LEDs are used in a circular configuration in order to improve the range that the IR blast can travel when triggered. The camera was mounted at an elevated position for two reasons. One, to prevent damage to the camera in the event that a car would be driven into an obstacle. Second, to give the user a better perspective of where they need to drive their car in order to get ot the next gate.

2. *Analog Design*: Two circuits were designed using analog components to achieve the specifications needed for this project. These circuits were designed to minimize power consumption, maximize range of use and facilitate detection. With these design constraints in mind, the transmitter circuit shown in Fig. 7 makes use a resistor-capacitor configuration (resistors R1 and R2 and capacitor C1) to set the frequency of the NE555 Timer chip to 1.38kHz. This timer will drive a pulsing signal to T1. When the IR software drives T4 with a logic high, the pulsing signal is allowed to propagate to the seven IR LEDs, allowing them to blast a signal forward. C2 is simply used as a decoupling capacitor. The IR receiver circuit shown in Fig 8 makes use of a double gated input system to reduce power draw and act as a low-cost analog to digital converter by driving input signals to power rail voltages. The double-gated system allows power to only be drawn when the receiver photodiode is triggered, thus minimizing power consumption.

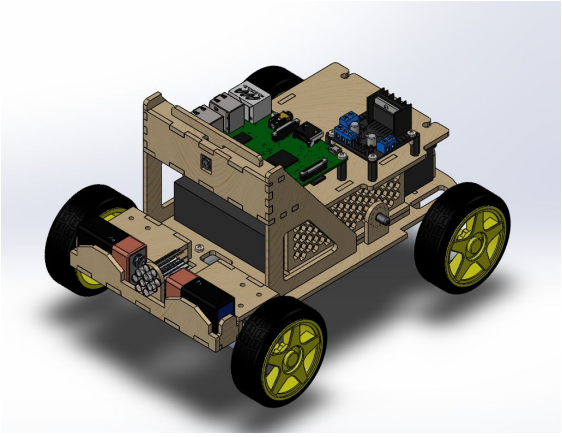


Fig. 7. 3D model for the frame of the car.

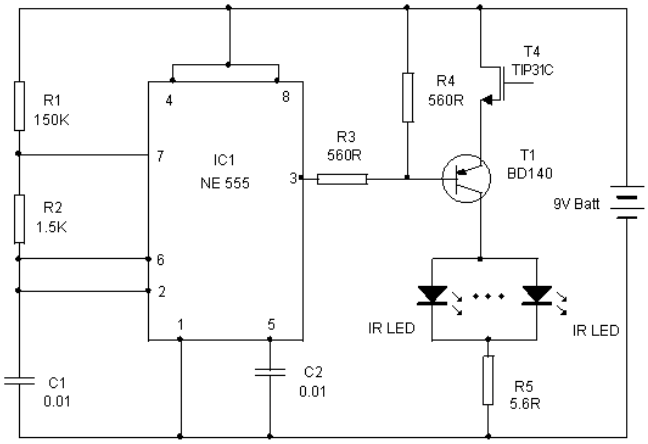


Fig. 8. Circuit diagram for IR blaster

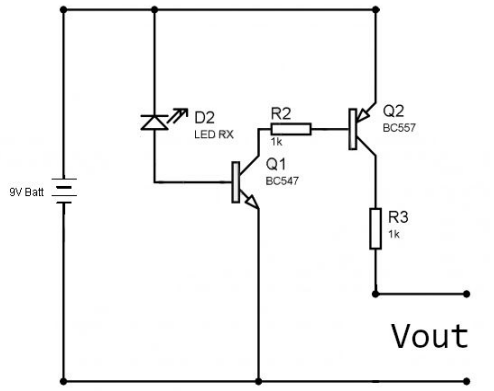


Fig. 9. Circuit Diagram for IR receiver

VI. Project Management

A. Schedule

Refer to the end of the report for a full schedule.

B. Team Member Responsibilities

Broadly speaking, David was responsible for software related to the game and PCs, Sourav was responsible for software related to the Pi and its hardware, and Bujji was responsible for hardware including the physical design of the cars and circuit design, though of course we will work together when we begin integrating our components. Refer to the attached schedule for a detailed breakdown of tasks and responsibilities—David's tasks are in yellow, Sourav's in blue, and Bujji's in green.

C. Budget

Refer to end of report for budget details and bill of materials.

D. Risk Management

1. Camera Turbulence: It was discovered that video feed stream from a small vehicle often suffers from micro-jitters. After investigating into solutions to this problem, we were unable to find affordable hardware solutions and effective real-time software to repair the blur seen in the video feed. To mitigate this risk, we designed a low-cost camera housing with rubber bands for suspension. However, in our final design we replaced the housing with a simpler one because camera turbulence was not as much of an issue as we had imagined.
2. Video-Streaming Latency and Jitter: A possible risk with our project was the latency in the stream of the video and frame rate jitter. In particular, this risk was exacerbated by conditions of high network traffic on the router the cars and PCs were connected to. At Professor Mai's suggestion, we requested a router from Professor Nace to use a LAN connection with our devices only instead of CMU's network, which greatly reduced frame drops in the video stream.
3. The Sun: In areas with too much sunlight or other bright lights, the IR receivers could be falsely triggered without a corresponding item use, causing players to be falsely hit. Our solution to this was to wrap the IR so they would only be triggered by IR blasts directed at them. This was only somewhat effective, but luckily the demo space in Weigand gym had sufficiently diffuse lighting to not activate the receivers.
4. Budget: Since our project had a wide variety of physical components, the budget was a big risk factor. Although our budget for all three cars and all the gates fit within the $600 allotted, we were forced by our limited budget to purchase parts that did not meet our requirements, namely the RFID readers. To mitigate this risk, we originally planned to purchase only two cars' worth of parts, giving us around $200 of slack to account for any problems that may have arisen with our parts, but we eventually opted to build all three cars instead of faster RFID readers.
5. Personal schedules: Since two of the members of our team are booth chairs, we scheduled our working weeks around build-week. This meant that there were two weeks (spring break and build-week) that members of our team were not able to work. Our original schedule left slack during those weeks, but we eventually fell behind schedule nonetheless and had to redistribute tasks between us each week to whomever had fewer assignments due at the time.

VII. Related Work

Though we could not find a product in the marketplace that approached the issue of AR in physical games quite like ARioKart, there were several projects that gave us inspiration when we were brainstorming.

Chief amongst these was Anki Overdrive[2]. Anki Overdrive is a low-power bluetooth application that networks phones with physical HotWheels-esque cars. The cars themselves have AI that drive the car in a portable map, as a circuit mat that can be laid down. The inspiration was having real mobile vehicles networked into a game that was controlled through a general purpose computing device like a phone or a laptop. Anki Overdrive diverges technically from our project in that it lacks an AR component, and the cars are not controlled directly by the user but through AI.

A more similar product which we researched was Hado Kart[3], an augmented reality version of the game of Mario Kart that allows people to be seated in the kart they drive. This game, while having features from Mario Kart like items and coins, was not actually a race. Players compete to collect as many coins as they can in an arena of sorts. The product more brings Mario Kart into the world of bumper cars, than bumper cars into the world of Mario Kart.

VIII. Summary

For the most part, our system met our most important requirements. ARioKart is a fun gaming experience that can be used for hours. Users enjoy playing a multiplayer game

with their friends where they get to drive physical cars over a completely customizable track. However, while ARioKart is an entertaining gaming system, it does have a few issues.

One of the biggest problems our cars have is that they aren't very easy to drive. While implementing differential steering was easier, and saved us from dealing with a lot of mechanical challenges, our cars suffer in their ability to turn. If we had more time, an obvious solution would to have simply chosen rear-wheel drive for the cars, with front wheel Ackerman steering.

Additionally, the limited performance of the RFID scanner is a great hindrance to users during gameplay. Originally we had thought that we would be able to scan gates from the cars while they drove at top speed. We tried two different RFID readers and two different types of RFID tags, yet were unable to meet this specification. The cars need to be slowed down to a "scanning speed" to be able to reliably detect the gates as they drive over them. If we had more money allocated to us in our budget, we would've been able to purchase a high speed RFID reader, or if we have more time, we could've designed our own high speed RFID reader.

A. *Future work*

If we were to continue work on this project, more time alone would not be sufficient. Many of the issues we ran into over the course of the semester were due to budget constraints. As such, to truly keep working on our project we would need to have an expanded budget. Given such, there are a few important improvements we have already pinpointed.

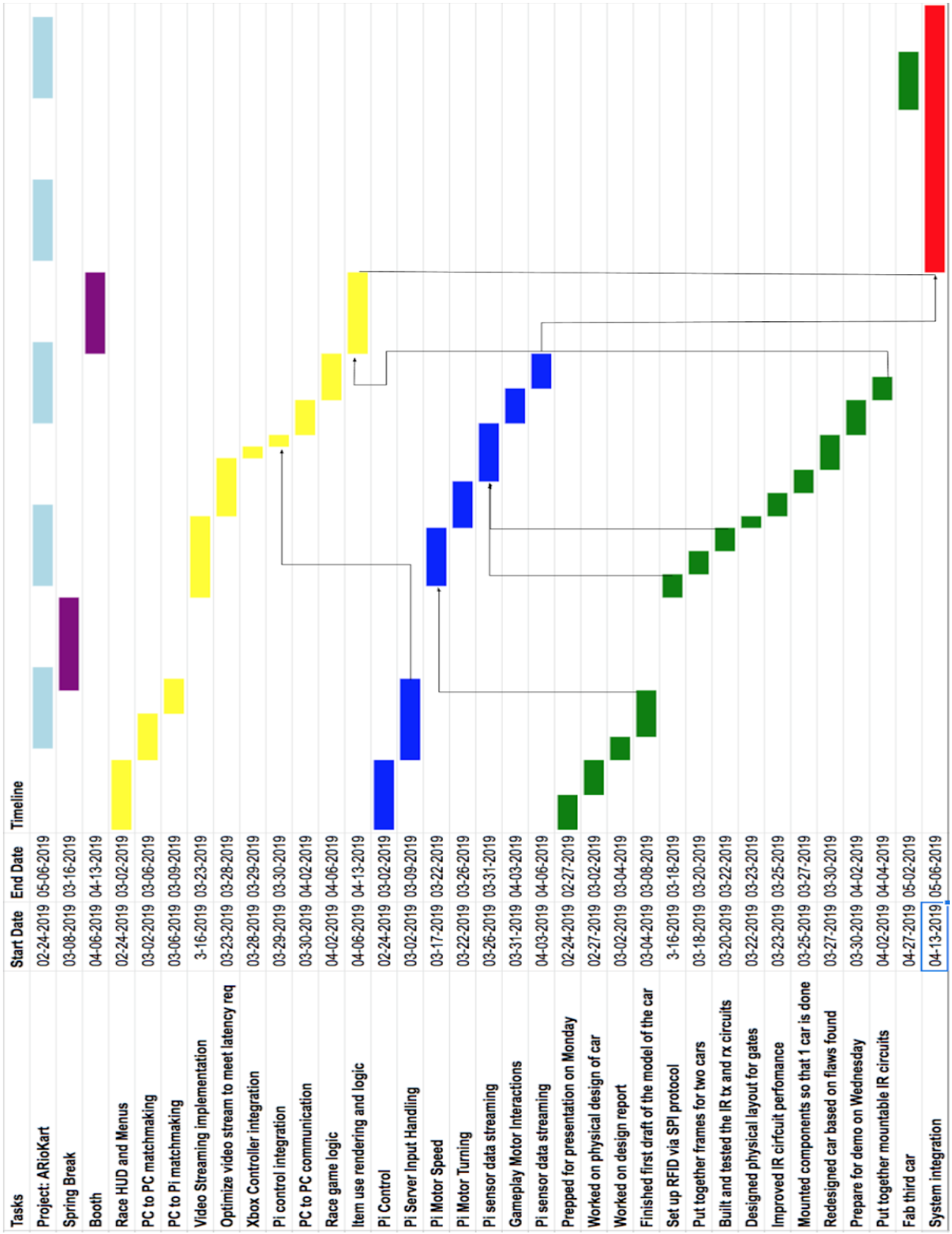
1. PCBs: The cars could very much benefit from PCBs, which would reduce clutter of circuits, reducing the possibility of wiring disconnects due to use. The PCB would also be valuable in ensuring the replicability of the IR circuits. As all three would have the same design, there would be no variation, as opposed to the current method in which errors in soldering could cause differences between each car.
2. Ackermann Steering - The steering apparatus of the car, as previously mentioned, currently utilizes torque vectoring and differential steering. This form of steering, while widely used, is almost always supplemental to Ackermann steering, which provides finer control of direction. This would involve a slight re-design of the car to accommodate the physical implementation of Ackermann Steering, as well as a complete rework of the steering software.
3. Data Transfer Through IR: Gameplay interactions could be enhanced if we were able to send data via the IR blaster, and receive that data on the various receivers on the cars. This would've also helped mitigate the issues with ambient sunlight in the demo area, and maybe our IR system more robust.

B. *Lessons Learned*

- *Never believe the posted hardware specs*
  Several of the hardware parts that we purchased for our project did not meet the hardware specifications listed on the website. This applied to our motors and RFID scanners. We'd recommend that students buy hardware parts that surpass the minimum specification that they desire.
- *Plan for setbacks*
  While wrote and cliche, this advice is really the most important. Several times in our project we were forced to go back and redo an entire part as it did not perform as expected. It is important to allocate the time for things like this in your schedule.
- *Choose a project within budget*
  A lot of sacrifices in our project were made so that we could have the budget to manufacture three cars. We recommend that students research the price of all of the parts so they can better understand what they can really do with the budget allocated with them.

References

[1] Techopedia. "What is Augmented Reality?", https://www.techopedia.com/definition/4776/augmented-reality-ar
[2] Anki. "Anki OVERDRIVE - Intelligent Racing Robot System", https://anki.com/en-us/overdrive.html
[3] Hado Asia. "HADO - Augmented Reality Techno Sports in Signapore", https://www.hado-asia.com/games/hado-kart/



Tasks	Start Date	End Date	Timeline
Project: ARioKart	02-24-2019	05-06-2019	
Spring Break	03-08-2019	03-16-2019	
Booth	04-06-2019	04-13-2019	
Race HUD and Menus	02-24-2019	03-02-2019	
PC to PC matchmaking	03-02-2019	03-06-2019	
PC to Pi matchmaking	03-06-2019	03-09-2019	
Video Streaming implementation	3-16-2019	03-23-2019	
Optimize video stream to meet latency req	03-23-2019	03-28-2019	
Xbox Controller integration	03-28-2019	03-29-2019	
Pi control integration	03-29-2019	03-30-2019	
PC to PC communication	03-30-2019	04-02-2019	
Race game logic	04-02-2019	04-06-2019	
Item use rendering and logic	04-06-2019	04-13-2019	
Pi Control	02-24-2019	03-02-2019	
Pi Server Input Handling	03-02-2019	03-09-2019	
Pi Motor Speed	03-17-2019	03-22-2019	
Pi Motor Turning	03-22-2019	03-26-2019	
Pi sensor data streaming	03-26-2019	03-31-2019	
Gameplay Motor Interactions	03-31-2019	04-03-2019	
Pi sensor data streaming	04-03-2019	04-06-2019	
Prepped for presentation on Monday	02-24-2019	02-27-2019	
Worked on physical design of car	02-27-2019	03-02-2019	
Worked on design report	03-02-2019	03-04-2019	
Finished first draft of the model of the car	03-04-2019	03-08-2019	
Set up RFID via SPI protocol	3-16-2019	03-18-2019	
Put together frames for two cars	03-18-2019	03-20-2019	
Built and tested the IR tx and rx circuits	03-20-2019	03-22-2019	
Designed physical layout for gates	03-22-2019	03-23-2019	
Improved IR cirfcuit perfomance	03-23-2019	03-25-2019	
Mounted components so that 1 car is done	03-25-2019	03-27-2019	
Redesigned car based on flaws found	03-27-2019	03-30-2019	
Prepare for demo on Wednesday	03-30-2019	04-02-2019	
Put together mountable IR circuits	04-02-2019	04-04-2019	
Fab third car	04-27-2019	05-02-2019	
System integration	04-13-2019	05-06-2019	

	No.	Part	Note	Dat	Supplier	Link	Price per piece	No. per car	Price per car	No. per project	Total Cost	Shipping	Category Cost	Car Cost
Control	1	Raspberry pi 3 B+			Amazon (Prime eligible)	https	$38.20	1	$38.20	3	$114.60	$0.00	$369.07	$597.91
	2	MicroSD Card (16GI	class 10 c		Microcenter	https:	$3.49	1	$3.49	3	$10.47	$0.00		
	3	Camera Pi v2	Has usefu		Microcenter	https	$24.99	1	$24.99	3	$74.97	$0.00		
	4	Motors	Motor(6(		Amazon (Prime eligible)	https	$18.58	2	$37.16	6	$111.48	$47.67		
	5	Motor Drivers	Pack of 5		Amazon (Prime eligible)	https	$9.88	1	$9.88	1	$9.88	$0.00		
Structure	6	Wheels (4 pieces)	2.56 inch		Amazon (Prime eligible)	https	$10.19	1	$10.19	3	$30.57	$0.00	$60.56	
	7	Axle (10 pack)	200mm x		Amazon (Prime eligible)	https	$14.99	1	$14.99	1	$14.99	$0.00		
	8	Wood	For maki		Makerspace		$3.00	1	$3.00	5	$15.00	$0.00		
Power	9	Portable Battery (6000mAh)			Amazon (Prime eligible)	https	$33.99	1	$33.99	3	$101.97	$0.00	$101.97	
RFID	10	RFID module			Amazon (Prime eligible)	https	$9.99	1	$9.99	2	$19.98	$0.00	$48.74	
	11	RFID reader (better)			Amazon (Prime eligible)	https	$10.99	1	$10.99	1	$10.99	$0.00		
	11	NFC Stickers	Pack of 5		Amazon (Prime eligible)	https	$17.77	1	$17.77	1	$17.77	$0.00		
IR	12	IR Emitter and rece	Pack of 5		Amazon (Prime eligible)	https	$10.99	1	$10.99	1	$10.99	$0.00	$17.57	
	13	Timer Chip	Pack of 3		Amazon (Prime eligible)	https	$6.58	1	$6.58	1	$6.58	$0.00		