

Augmented Reality Capture the Flag

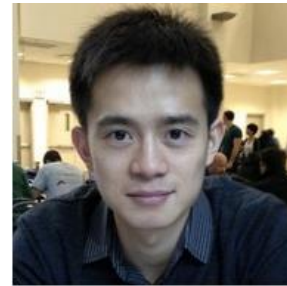
Group Members:



Amshu Gongal



Kenan Kigunda



Matthew Pon

Project Summary:

This project is an augmented reality capture-the-flag game, in which players control RC cars through a web interface, with the objective of being the first player to reach the opponent's flag and return to his own base. Infrared will be used to implement a shooting feature, and a rover will temporarily freeze when shot.

Group Number: 7

Table of Contents

1	Abstract.....	5
2	Functional Requirements.....	5
3	Design and Description of Operation.....	6
3.1	Design Overview	6
3.2	Network Communications	7
3.3	Wi-Fi Hardware	9
3.4	Infrared Hardware	11
3.5	Motor Hardware	13
4	Bill of Materials	15
5	Datasheet.....	18
5.1	System Block Diagram.....	18
5.2	Speed and Performance Requirement:	18
5.3	Power Calculations:.....	18
5.4	Operating Conditions:.....	19
5.5	I/O Signals	19
5.5.1	Off-Board to Board:.....	19
5.5.2	Off-Board electronics to World.....	20
6	Background Reading	21
6.1	WebSockets.....	21
6.2	Augmented Reality Universal Controller and Identifier (ARUCI)	21
6.3	AR Shooter: An Augmented Reality Shooting Game System.....	21
6.4	Data Formats for IR Remote Control	21
7	Software Design	22
7.1	Rover Software.....	22
7.1.1	Wi-Fi Handler Start Task.....	22
7.1.2	Infrared Receiver Update Task.....	22
7.1.3	Network Sender Update Task	23
7.1.4	Network Receiver Ping Task.....	23
7.1.5	Network Receiver Update Task.....	23
7.1.6	Infrared Sender Update Task	23

7.1.7	Motor Handler Update Task	24
7.2	Server Software.....	24
7.3	Player Software	26
8	Test Plan.....	27
8.1	Software Test Plan	27
8.1.1	Command Interface	27
8.1.2	Network Interface	27
8.1.3	Infrared Interface	28
8.2	Hardware Test Plan.....	28
8.2.1	Infrared Emitter And Receiver	28
8.2.2	Xbee Wi-Fi Module.....	29
8.2.3	Motor Controller	29
8.3	Integration Test Procedure:.....	29
9	Results and Characterization	30
9.1	Infrared Readings.....	30
9.1.1	Ambient infrared readings	30
9.1.2	Infrared readings vs distance	30
10	Safety & Sustainability	31
10.1	Wi-Fi Connection Failure.....	31
10.2	Speed of Vehicle.....	31
10.3	Voltage, Current, & Power	31
10.3.1	Motor Component	31
10.3.2	Wi-Fi Component	31
10.3.3	Infrared Circuitry	31
10.3.4	Total Circuit Power.....	32
10.4	Energy	32
10.4.1	Kinetic Energy.....	32
10.4.2	Stored/Potential Energy.....	32
10.5	Summary of Safety Concerns	32
10.6	Energy Cost	32
10.7	C02 Production from an Equivalent Coal Generation of Energy	32
10.8	Solar Cell Area	32

11	Environmental Impact.....	33
11.1	Hazardous Substances/ RoHS Compliance	33
11.2	Other Environmental Points	33
12	References	34
13	Appendix A Quick Start Manual	35
14	Appendix B Future Features:	36
14.1	Future Feature 1:	36
14.2	Future Feature 2:	36
14.3	Future Feature 3:	36
14.3.1	Details of Operation with GPIO Camera	36
14.3.2	Details of Operation with IP Camera	37
15	Appendix C Hardware Documentation	38
15.1	Top-Level.....	38
15.2	Wi-Fi	39
15.3	Infrared	40
15.3.1	Emitters.....	40
15.3.2	Receivers	41
15.3.3	Motor	42
16	Appendix D Source Code.....	43
16.1	Index to Source Code	43
16.2	Data Flow	44

1 Abstract

The goal of this design project was to create an augmented-reality capture the flag game with two RC vehicles. Each vehicle contains an on-board Altera DE0 Nano FPGA board, an infrared emitter circuit using the LTE-4208, 4 infrared receiver circuits using the LTR-3208E, an XBee S6 Wi-Fi module, and a Polulu Low-Voltage Motor Controller. 7 AA batteries are used on each rover; four batteries power the DE0 Nano board, which in turn power the Wi-Fi module and the infrared circuitry, and three batteries power the motor controller and the motor. On-board components communicate with the players through a web server and through HTTP. The infrared module was interfaced to the Wi-Fi module and server to allow for the shooting of opponents. Users log onto a web server to control vehicle movement. Players will be able to control the speed of each track (right or left), and a skid steer mechanism will be used to control rover movement. The infrared circuitry, composed of 2 infrared receivers on each of the 4 sides, and 3 infrared emitters on the front of each rover, will allow for simulated 'shooting' of opponents. Overall the project was definite success, and almost all intended functionality was properly implemented in the final product.

2 Functional Requirements

This project is an augmented-reality Capture the Flag game. Two players each log onto a browser and use a simple user interface to control RC cars. Functionality included the control of left and right track speed, as well as infrared transmission. Independent track controls allowed for finer movement of the vehicles.

In order to make the game description easier to understand, let us first define a few terms:

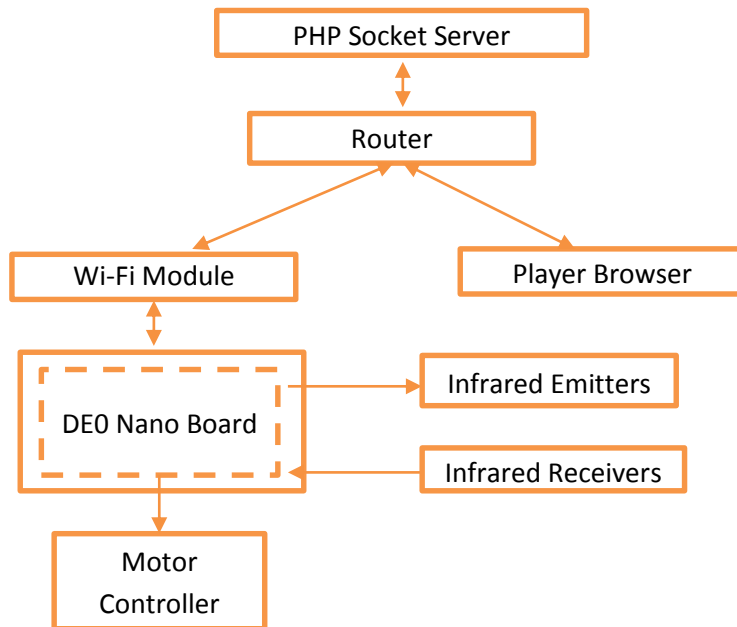
Base	A location for each player, known by all players at the start of the game
Game Space	Any open location (indoors) in which the players choose to play the game
Flag	An object placed at each player's base
Capture the Flag	To reach the opponent's flag. (Determined by a referee)
Carrying the Flag	Having reached the opponent's base but not yet returned home
Return home	To return to one's base after having captured the flag
Shoot	To register a shot through infrared transmission/reception

In the beginning state of the game, each car will be in its own base. The first player to navigate his vehicle to his opponent's base, capture the flag, and then return home, will win. Infrared reception at the home bases and enemy flags will be used to avoid the need for location sensing. Each rover contains 1 DE0 Nano board, 1 Polulu low voltage motor controller, 1 XBee S6 Wi-Fi module, 8 infrared receivers, and 3 infrared emitters.

The next feature added was the ability for users to shoot each other. Once a shot is registered, the car that was shot freezes for a set period of time (we implemented this as 6 seconds). Cars are able to be shot from a range of approximately 1-2 feet.

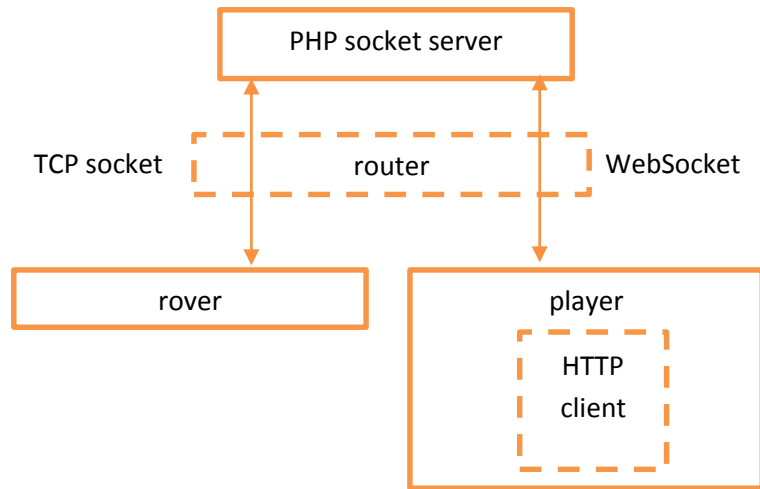
3 Design and Description of Operation

3.1 Design Overview

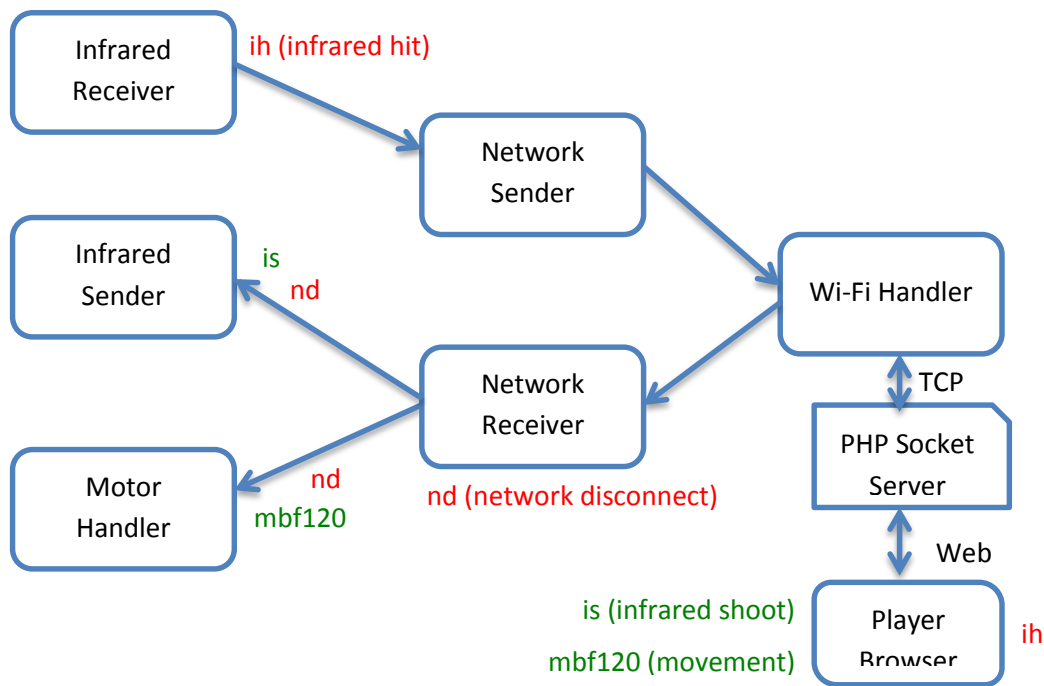


On each rover, there are four main components interfaced to the DE0 Nano board: a motor controller, which is connected to a pair of motors which move the rover through skid steering; a set of infrared emitters, which are used to shoot opponents; a set of infrared receivers, which are used to detect infrared shoot events; and a Wi-Fi module, which is used to communicate with the rest of the world. By joining a local area network provided by a router, the rovers can connect to a PHP socket server, while players on the same network can connect to the server using any HTML5 browser. The server keeps track of the game state and relays messages between players and servers, while client-side JavaScript provides the user interface used to control the rovers and mark infrared hits.

3.2 Network Communications



For network communications between the players and the rovers, we set up a local area network using a router. The network is encrypted using WPA2, and all network communications flow through a PHP socket server. Each rover connects to the PHP server using a TCP socket that is provided by the firmware on the Xbee Wi-Fi module, while each player connects to the server using the [WebSocket](#) protocol specified in HTML5. The server relays messages from a player to the corresponding rover, and relays events such as infrared hit events from the affect rover to all players, creating two-way communication between the rovers and the players that allows the players to control their rovers and be aware of the state of each rover.



Network events are created by both the rovers (shown in red) and the players (shown in green). Each event is a short string terminated by a newline. The first character of the event marks the type of the event: i for infrared, m for motor, and n for network. The remaining characters give the parameters of the event: is is used to mark an infrared shoot event; ih, an infrared hit event, and m\$m\$d\$sss, a motor movement command, where \$m indicates which motor to move (one of l “left”, r “right”, or b “both”), \$d indicates the direction (one of f “forward” or b “back”), and \$sss indicates the speed (between 0 “stop” and 127 “max”).

On the rover side, the InfraredReceiver class polls the analog-to-digital converter for the level of the infrared receivers. When the level exceeds the infrared hit threshold, an infrared hit event ih is generated. This event is forwarded to the NetworkSender class over a queue. The network sender reads events from its queue and uses the WifiHandler class, which provides a high-level view of the interfacing to the Xbee Wi-Fi module, to send the hit event to the server, which then forwards the hit event to all the players.

On the client side, there is an HTML interface with SVG objects that have JavaScript callbacks attached to the click event. When the user clicks on one of these UI buttons, the callback sends the corresponding message to the server using the WebSocket. The server associates each WebSocket connection with a particular rover, and forwards commands received on a WebSocket to the corresponding rover.

The NetworkReceiver class uses WifiHandler to poll for incoming network events, and forwards the events over a queue to the listener class corresponding to the type given by the first character of the

event. Infrared shoot commands are executed by the InfraredSender class, while motor movement commands are executed by the MotorHandler class.

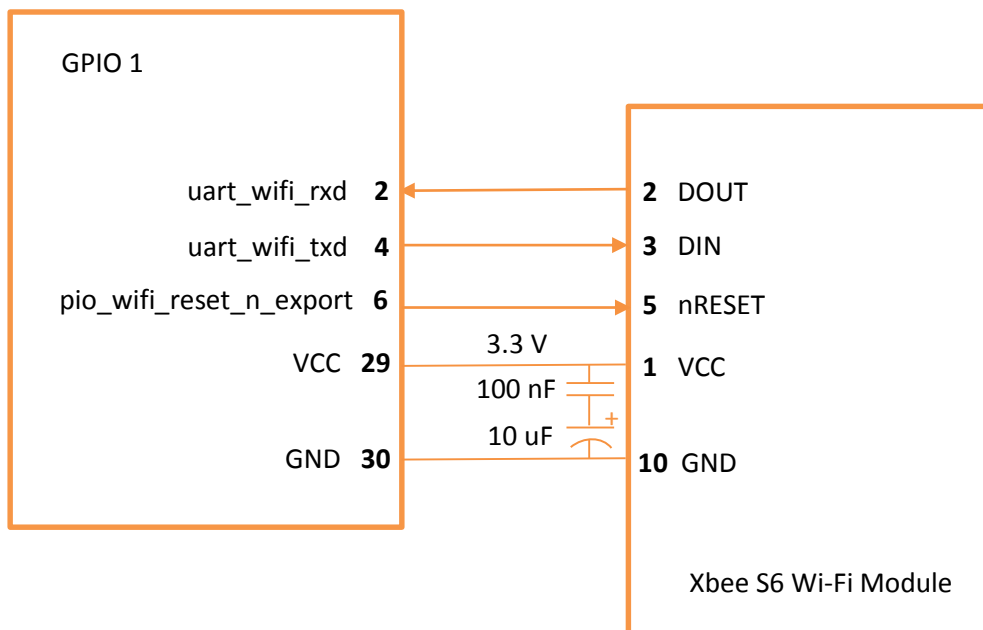
In addition to checking for new messages from the server, Network Receiver also pings the server periodically to keep the TCP connection alive. The Xbee Wi-Fi module has a configurable timeout that we set to the maximum of 25.5 seconds. Network Receiver pings the server every 15 seconds and displays the connection status on the LED. If the ping fails, the receiver issues a nd “network disconnect” command to all listeners, which react accordingly. Namely, MotorHandler stops the motors on network disconnect.

The WifiHandler class is synchronized, meaning that only one caller can read from or write to the server at a time. When the Wi-Fi connection is in use, callers will wait up to a timeout fixed at around one second to use the connection. If the timeout expires, the caller will try again after the task delays, allowing other tasks to run.

In addition to the aforementioned messages, there are two types of network control messages. Messages starting with a plus + are network connect messages. They are the first message sent to the server and identify the type and ID of the client; for example +r1 indicates a new connection from rover 1, while +p1 indicates a new connection from player 1. The ID is how the server associates players with rovers; messages from player 1 will be forwarded to rover 1.

Messages starting with a colon : are network status messages. The server sends back :ok to any message that was received and processed successfully, and :err to any message that could not be processed; for example, if the first message sent to the server on a new connection is not a network connect message with a type and ID, then the server will return :err.

3.3 Wi-Fi Hardware



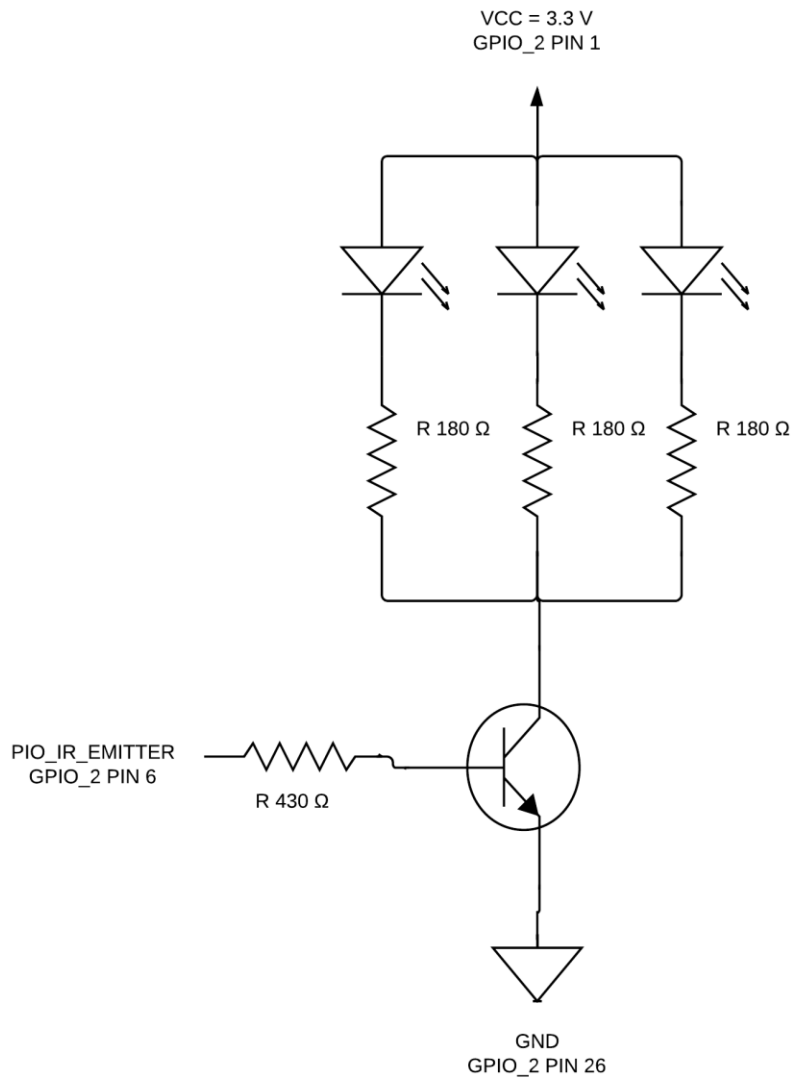
Wi-Fi communication with the rovers is achieved through the use of a Xbee S6 Wi-Fi module. The board communicates with the Wi-Fi module using RS232 UART. Outgoing serial data is transmitted from GPIO 1 pin 2 to Xbee pin 2, and incoming serial data is transmitted from Xbee pin 3 to GPIO 1 pin 4. The GPIO pins are mapped to an *altera_up_avalon_rs232* Qsys component name *uart_wifi*, which handles the implementation of RS232 UART and is memory mapped to `UART_WIFI_BASE`. Any data written to `UART_WIFI_BASE` is transmitted through *uart_wifi_txd* to the Wi-Fi module, and data sent by the module to *uart_wifi_rxd* can be read through `UART_WIFI_BASE`. The *altera_up_avalon_rs232* Qsys component also contains first-in-first-out (FIFO) queues for increased flexibility, and Alterra provides an API for checking the status of the FIFO queues and then reading and writing, which is used instead of reading and writing to `UART_WIFI_BASE` directly.

The Xbee Wi-Fi module is used in transparent mode, which means that any data sent over UART is forwarded to the network, and any data received from the network is forwarded over UART. The module is first configured by entering command mode and sending three plus characters `+++`. We then use AT commands to configure the module to use the TCP protocol and establish a connection to our server (more details in Software Design). Then, when the Wi-Fi module is not in command mode, the first data it receives over UART will cause a TCP socket to be opened targeting the server. Once the socket is opened, all data transmitted to the Wi-Fi module is forwarded over that socket, and all data received on the socket is forwarded to the board.

The Wi-Fi module has a reset input on pin 5 which is connected to the PIO component *pio_wifi_reset_n* on GPIO 1 pin 6. This reset input is active low, and so is set to one when the board starts up and remains high.

The Wi-Fi module is powered from the board's logic power supply at 3.3 V. To reduce noise, particularly from the motors, there is an electrolytic capacitor of 10 μ F and a ceramic capacitor of 100 nF between the Wi-Fi power and ground lines.

3.4 Infrared Hardware

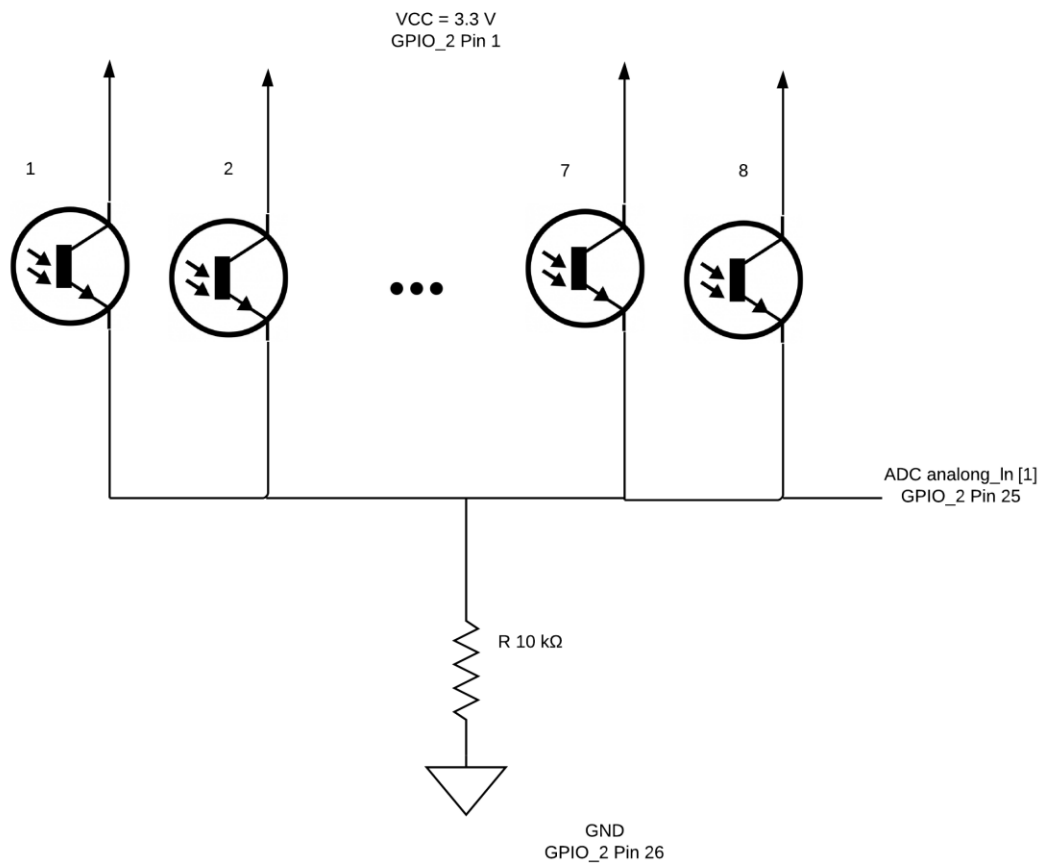


The infrared emitters are LTE 4208 photodiodes. To prolong the lifetime of the emitters, we operate them near the lower end of their current range up to 50 mA, limiting the forward current to $I_D = 10$ mA with a corresponding forward voltage of around $V_D = 1.2$ V. To limit the forward current, each diode has a current-limiting resistor R_D with value calculated as

$$\begin{aligned} R_D &= \frac{V_{CC} - V_D - V_{collector}}{I_D} \\ &= \frac{(3.3 - 1.2 - 0.2) V}{10 mA} \\ &= 190 \Omega \end{aligned}$$

The closest resistors available in the lab were $180\ \Omega$ and $200\ \Omega$. Since we were at the low end of the range and had room to grow upwards, we chose a limiting resistor of $180\ \Omega$, allowing for a current at or slightly above $10\ \text{mA}$.

The infrared emitters are turned on and off through an NPN transistor controlled through GPIO 2 pin 6. This pin is wired to the base of the transistor, through a resistor $R_B = 430\ \Omega$ whose value is chosen such that the base current is around one-tenth the collector current and the transistor stays in saturation. When the GPIO pin is high, the transistor turns on and allows current to flow through the photodiodes; when the GPIO pin is low, the transistor turns off and no current can flow through the photodiodes. This GPIO pin is mapped to the Qsys component *pio_ir_emitter*, which is memory-mapped to `PIO_IR_EMITTER_BASE`. As a result, writing 1 to `PIO_IR_EMITTER_BASE` turns the emitters on, and writing 0 to `PIO_IR_EMITTER_BASE` turns them off.



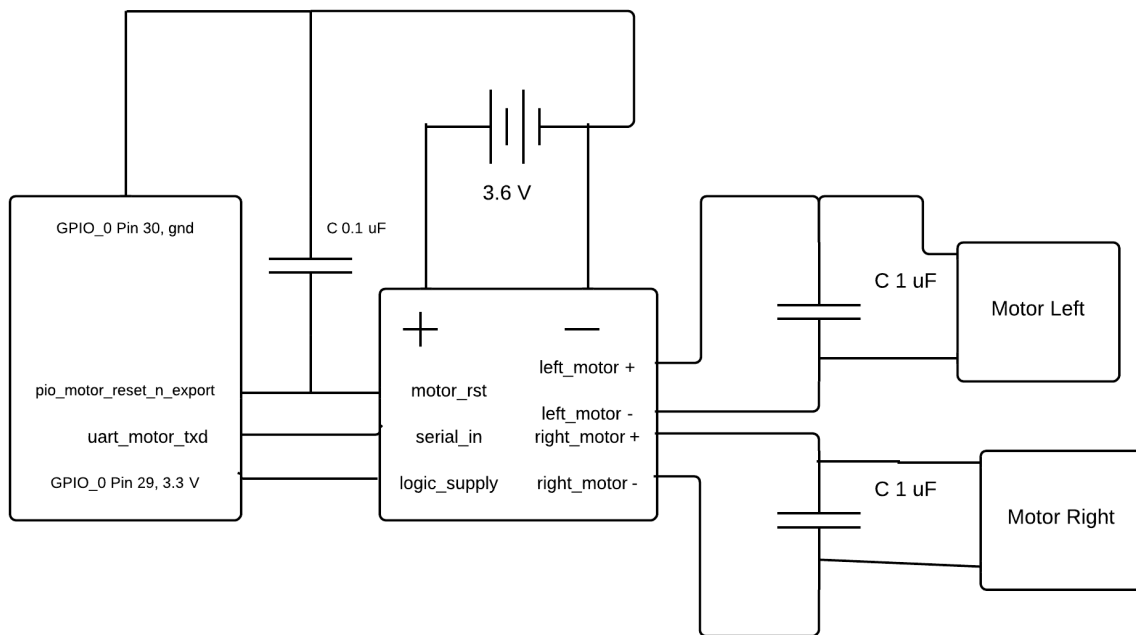
The infrared receivers are R3208E phototransistors. The amount of current flowing through these phototransistors is directly proportional to the intensity of the ambient light they receive. The collector of each receiver is connected to $3.3\ \text{V}$ power, while each emitter is connected to an $R = 10\ \text{k}\Omega$ pull-down resistor and the first analog input of the analog-to-digital converter (ADC) on the board. There are a total of eight receivers on each rover, with two on each side. When the level of infrared light received

by all the receivers is low, the pull-down resistor keeps the input on the ADC low. When any receiver receives significant infrared light, it turns on and pulls the input on the ADC towards power. The degree to which the ADC input is pulled towards VCC reflects the intensity of the ambient light.

Altera provides a Qsys component *altera_up_avalon_de0_nano_adc* for interfacing the ADC. The ADC has eight channels, of which we use only the first. In software, Alterra provides an API for reading the value of a particular channel, returning an eight-bit integer. This integer is compared to an infrared hit threshold, which was determined experimentally by finding the maximum reading caused by ambient light in the location of expected use.

3.5 Motor Hardware

We use Polulu Low Voltage Dual Serial Motor Controller to drive the motor. The motor controller uses RS-232 UART communication protocol to receive commands from the DE0 Nano board through the GPIO's. The board receives commands from the player and forwards the motor command to the motor controller through GPIO 0. The GPIO pins are mapped to an *altera_up_avalon_rs232* Qsys component name *uart_motor*, which handles the implementation of RS232 UART and is memory mapped to *UART_MOTOR_BASE*. Any data written to *UART_MOTOR_BASE* is transmitted through *uart_motor_txd* to the motor controller.



The motors are powered through a separate battery pack (3.6 V) on the rover. The motor controller is powered through the DE0 Nano board with a 3.3 V power supply. The motor controller has an reset input which is connected to the PIO component *pio_motor_reset_n* on GPIO 0 pin 33. This reset input is active low, and so is set to one when the board starts up and remains high.

The motors used are noisy so capacitors are installed along the terminals to reduce noise. The noise from the motors interferes with the reset line causing the motor to reset while in operation so 0.1 uF capacitors are installed between the motor_reset and ground to reduce noise. Also LC filters are installed between the power outputs from controller to the motors to reduce noise.

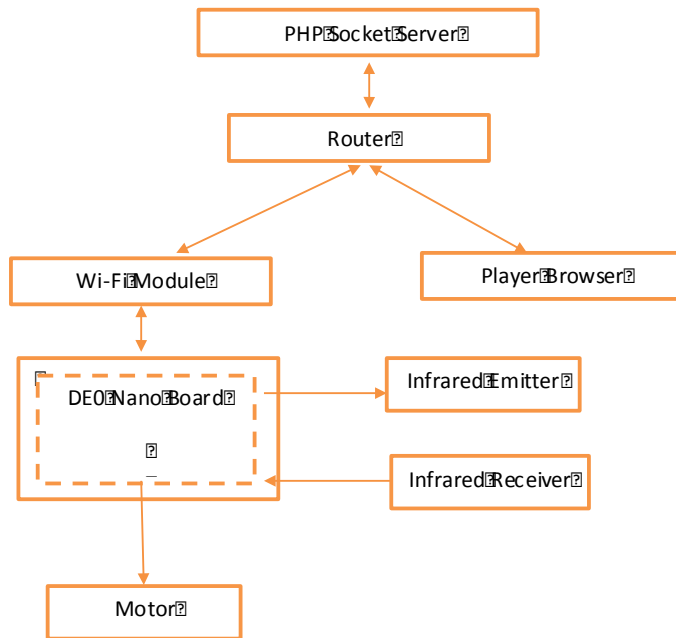
4 Bill of Materials

Part	Quantity	Unit Cost	Net Cost	Supplier	Link	Datasheet
Altera DE0 Nano FPGA board	2	\$59.00	\$118.00	Lab Supplied	http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html	ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE0-Nano/DE0_Nano_Datasheets.zip
Rover Platform RB-Tam-27	2	\$18.99	\$37.98	Robotshop	http://www.robotshop.com/ca/en/tracked-vehicle-chassis-kit.html	
Twin Motor Gear Box RB-Tam-01	2	\$10.88	\$21.76	Robotshop	http://www.robotshop.com/ca/en/tamiya-twin-motor-gearbox.html	
Xbee S6 Wi-Fi module with PCB antenna XB24-WFPIT-001	2	\$41.92	\$83.84	Digikey	http://www.digikey.ca/product-detail/en/XB24-WFPIT-001/602-1213-ND/2700943	http://ftp1.digi.com/support/documentation/90002124_F.pdf
Polulu Low Voltage Dual Serial Motor Controller	2	\$44.39	\$88.78	Robotshop	http://www.robotshop.com/ca/en/pololu-low-voltage-serial-controller.html	http://www.robots.com/media/files/pdf/smc05a-low-voltage-dual-serial-motor-controller-guide.pdf
Metal Standoffs	30	\$0.40	\$12.00	Lab Supplied		
Perf Boards	4	\$7.50	\$30.00	Lab Supplied		
Battery Holders (4 AA)	2	\$5.85	\$11.70	Lab Supplied		
Battery	2	\$5.85	\$11.70	Lab Supplied		

Holders (3 AA)						
Infrared Emitters	6	\$0.40	\$2.40	Lab Supplied		
Infrared Receivers	16	\$1.00	\$16.00	Lab Supplied		
LED	2	\$0.50	\$1.00	Lab Supplied		
Headers	2	\$8.00	\$16.00	Lab Supplied		
Misc. Consumables (Wires, solder, glue, velcro etc)			\$8.00	Lab Supplied		
Total			\$459.16			

5 Datasheet

5.1 System Block Diagram



The diagram represents our entire system. The rover consists of the DE0 Nano board, Wi-Fi module, Infrared emitters and Receivers and motor controller. We have a PHP socket server is the server side to control the state of the game and Player browser is the client side for user interface.

5.2 Speed and Performance Requirement:

Speed of the processor is not a major concern in this system as most of the computation are computed in the server side that controls the game state. The micro-controller parses the instruction from the server and relays either to the infrared emitter or motor controller. Other task of the FPGA is to continuously poll the infrared receiver to detect infrared emission. The network listener continuously polls for data from the server. None of the tasks require vast memory for operation.

5.3 Power Calculations:

The rover requires 7 NiMH batteries to operate. 4 batteries (4.8 V) are used to power the Altera DE0 Nano board. 3 batteries (3.6 V) are used to power the motors.

Following calculations illustrates the power consumption when the motors running in maximum load:

Motors + Motor Controller:

$$(2 * 4200 + 10) \text{ mA} * 3.6 \text{ V} / 1000 = 30.276 \text{ Wh}$$

$$3 \text{ Batteries: } 2300 \text{ mAh} * 3.6 \text{ V} / 1000 = 8.28 \text{ Wh}$$

Estimate run time = $8.28/30.276 = 16.4$ minutes

Following calculations illustrates the power consumptions by DE0 Nano board, IR circuit and Wi-Fi module:

On-board Electronics:

Wi-Fi module + IR circuits + DE0 board = 490 mA = 490mA

4 Batteries: 2300 mAh

Estimate run time = $2300 \text{ mAh}/490 \text{ mA} = 4.7$ hours

Following calculations illustrates power consumption when the rover is idle:

Motors consume no power when the rover is idle.

On-Board Electronics consume 330 mA when the rover is idle.

5.4 Operating Conditions:

It is suggested that the rover is operated indoors with minimum exposure to infrared that can cause unexpected hit event to be recognized by the infrared receivers since our infrared signal are not encoded. The rover should not be operated further than the operating range of the router for network communications. Low network signal strength can cause the rover to disconnect from the server.

5.5 I/O Signals

5.5.1 Off-Board to Board:

5.5.1.1 Xbee Wi-Fi module to DE0 Nano board

Signals preceded by *n* are active low.

Signal	Origin	Pin	Voltage	Destination	Pin	Description
VCC	GPIO, 3.3 V	11	3.3 V	Xbee	1	Power Supply from DE0 Nano to Xbee
GND	GPIO	12	0 V	Xbee	10	DE0 gnd to Xbee gnd
DOUT	Xbee	2	0 - 3.3 V	GPIO_1	2	Data received from server
DIN	GPIO_1	4	0 - 3.3 V	Xbee	3	Data transmitting to server from board
nRESET	GPIO_1	6	0 - 3.3 V	Xbee	5	Reset Line

5.5.1.2 IR emitter and receiver to infrared controller

Signal	Origin	Pin	Voltage	Destination	Description
VS	GPIO_2, 3.3 V	1	3.3 V	IR receiver, through 33 to 1k Ω resistor and < 0.1 μ F	Voltage supply for IR receivers

				capacitor	
GND	GPIO_2	26	0 V	IR receiver, IR emitter	Board gnd
OUT	IR receiver		0 - 3.3 V	ADC	
IN	GPIO_2	6	0 - 3.3 V	IR emitter diode	PIO, powered to turn on IR emitter

5.5.1.3 Polulu motor controller to motor controller

Signal	Origin	Pin	Voltage	Destination	Pin	Description
Serial Input	GPIO_0	17	0 - 3.3 V	Motor controller	4	Motor commands transmitted though RS-232 serial communication
Reset	GPIO_0	33	0 - 3.3 V	Motor controller	5	Motor reset
GND	GPIO_0	30	0 V	Motor controller	2	
VCC	GPIO_0, 3.3 V	29	3.3 V	Motor controller	3	Logic Supply

5.5.2 Off-Board electronics to World

5.5.2.1 Polulu motor controller to power and motors

Signal	Origin	Voltage	Destination	Pin	Description
VIN	Power pack, 3V	3 V	Motor controller	1	Power supply for motors
GND	Power pack	0 V	Motor controller	2	Connected to board gnd and motor controller gnd
Motor 0	Motor controller	3 V	Left motor	6,7	Left motor power supply
Motor 1	Motor controller	3 V	Right motor	8,9	Right motor power supply

6 Background Reading

6.1 WebSockets

- WebSockets is an API that facilitates interactive communication between a user's browser and a server. It allows for messages to be sent to a server and event-driven responses to be received without the requirement of polling.
- For our project, we need to set up some type of duplex communication between the Wi-Fi module and the remote server. The WebSockets API will allow us to implement this functionality on the client side.
- Links:
 - <http://www.websocket.org/index.html>
 - https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications
 - <http://dev.w3.org/html5/websockets/>

6.2 Augmented Reality Universal Controller and Identifier (ARUCI)

- ARUCI is a design project from some electrical engineering students last year at the University of Waterloo.
- The students implemented a universal remote control, interfacing a smart phone along with Wi-Fi/RF/Infrared modules.
- Our project involves some similar applications, and seeing some of the challenges and workarounds that they have implemented may aid us in the design and troubleshooting process.
- Links for the ARUCI project:
 - <http://elecceleator.com/our-fourth-year-design-project-aruci/>
 - http://elecceleator.com/fydp_aruci/aruci_slides_w_notes.pdf

6.3 AR Shooter: An Augmented Reality Shooting Game System

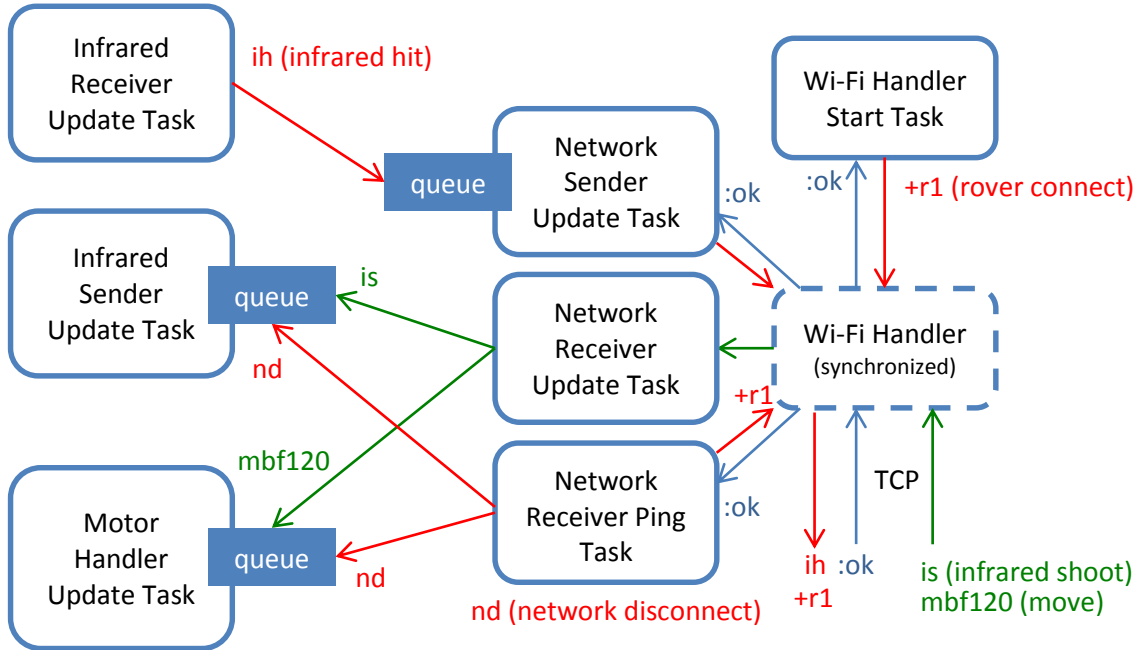
- This is proposed system for an augmented reality shooter that details some possibilities for creating such a system.
- The authors of the article tried LED markers for target shooting, but are also trying infrared patterns.
- This paper was a resource in deciding how to implement the shooting aspect of the game, since we removed cameras in the basic implementation.
- Link: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5643620>

6.4 Data Formats for IR Remote Control

- This article outlines the details of encoding data for infrared transmission.
- In order to deal with the abundance of noise in IR transmission/reception, we have to properly encode the signals. Different targets, along with flags and opponent vehicles should also be distinguished and encoded differently. This paper is a good resource in outlining the details of this process.
- Link: <http://www.vishay.com/docs/80071/dataform.pdf>

7 Software Design

7.1 Rover Software



The rover software involves a total of seven tasks, which communicate through queues. Each of the tasks is managed through a C++ singleton class, i.e. there is one object instantiated for each class. In general each class has an *update* task, with the exception of the NetworkReceiver class which has both an update and a connect task. There are two categories of classes: the datasource classes InfraredReceiver and NetworkReceiver poll periodically to check for updates; the listener classes InfraredSender, MotorHandler, and NetworkReceiver wait on queues for new events from datasource classes and process them as they arrive.

7.1.1 Wi-Fi Handler Start Task

The Wi-Fi handler start task configures the Xbee Wi-Fi module, if necessary, and then sends a network connect message +r1 or +r2 to the server. The Wi-Fi module stores its configuration details in non-volatile memory, so the configuration steps are only run when the settings are changed. While the WifiHandler class attempts to communicate with the server, it indicates the network status on the LED, which will blink until the connection is successful. The WifiHandler is synchronized, such that only one caller can use the Wi-Fi connection at any given time, and so the start task locks WifiHandler until an :ok message is received from the server. This task has the highest priority so it is guaranteed to run first. Once the :ok is received, the start task is delayed indefinitely.

7.1.2 Infrared Receiver Update Task

The infrared receiver update task polls the analog-to-digital converter (ADC) for the reading from the infrared phototransistors. The InfraredReceiver class uses an alt_up_de0_nano_adc_dev to read the

ADC value, which is first shifted right by four bits because the lower four bits are not used by the ADC, and then subtracted from a base value, returning a reading between 0 and 255 corresponding the range of 0 and 3.3 V. This reading is then compared to `INFRARED_RECEIVER_HIT_THRESHOLD = 25`, which was experimentally determined to filter out ambient noise. If the reading is larger than the threshold, then the `InfraredReceiver` posts an infrared hit event `ih` to its listener queue, which is set at startup to the `NetworkSender` queue. Typically, the task runs every he tasks runs every `INFRARED_RECEIVER_NO_HIT_UPDATE_TIME_MILLIS = 200` milliseconds. However, when there is an infrared hit, the tasks is delayed for `INFRARED_RECEIVER_HIT_UPDATE_TIME_SECONDS = 4` seconds, so that any high infrared readings that occur within four seconds are counted as one infrared hit event.

7.1.3 Network Sender Update Task

The network sender update task pends on a listener queue for outbound network events such as infrared hits `ih`. When a new outbound event is posted to the queue, the `NetworkSender` class uses the `WifiHandler` class to send the event to the server, and waits for an `:ok` response. If the connection is busy, then the `NetworkSender` will wait until it is free to use. The `NetworkSender` uses the `WifiHandler` `sendAndConfirm` method, where the message will be resent if an `:ok` response is not received before the response timeout.

7.1.4 Network Receiver Ping Task

The network receiver ping tasks is similar to the Wi-Fi Handler start task. It sends a network connect message `+r1` or `+r2` to the server and expects an `:ok` response. If the `:ok` is not received before the response timeout, then the `NetworkReceiver` posts a network disconnect event `nd` to all of its listeners and indicates the network disconnect status on the LED.

7.1.5 Network Receiver Update Task

The network receiver update task polls the `WifiHandler` for new network data using the `hasData` method, which checks the read first-in-first-out buffer on the Wi-Fi UART line. If is new network data, the `NetworkReceiver` reads it using the `WifiHandler` `tcpReceive` method, which returns the characters sent from the server up to and including the next newline, or `NULL` if the Wi-Fi connection is busy. If the connection is busy, the `NetworkReceiver` gives up and retries on the next polling cycle; if a line is read, it looks at the first character of the line, which indicates the event type. Listeners are registered with the `NetworkReceiver` instance for a particular type: `InfraredSender` is registered under `i` while `MotorHandler` is registered under `m`. The `NetworkReceiver` will forward an event to the corresponding listener, or will drop the event if there is no matching listener. Because the events are transferred across queues between tasks and to allow for flexibility in the number of active events, network events are allocated dynamically, and must be freed by the listener.

7.1.6 Infrared Sender Update Task

The infrared sender update task pends on a listener queue for infrared shoot commands `is`. When a shoot command is posted to the queue, the `InfraredSender` class turns on the infrared photodiodes by writing one to `PIO_IR_EMITTER_BASE`. The diodes remain on for

INFRARED_SENDER_ON_TIME_SECONDS = 3 seconds, at which point the InfraredSender clears PIO_IR_EMITTER_BASE.

7.1.7 Motor Handler Update Task

The motor handler update task pends on a listener queue for motor movement commands. The first character of the command indicates the command type: m, for motor. The second character indicates which motor to move: l, for left, r, for right, or b, for both. The third character indicates the direction: f, for forward, or b, for backward. The remaining characters up to the newline indicate the speed, ranging between 0, for stop, and 127, for maximum speed. The MotorHandler class parses the movement command by comparing the first three characters and using `strtoul` to get the speed starting at the fourth character, and then generates the sequence of bytes to send to the motor controller over UART:

1. The first byte is the start byte 0x80. This byte is distinguished by having a most significant byte (MSB) of one; all other bytes have an MSB of zero.
2. The second byte is the device type, which for our motor controller is 0x00.
3. The third byte indicates both which motor and which direction, and is found by multiplying the motor number by two and adding one if the direction is forward. Since our left motor is mapped to motor 2, 0x04 indicates motor left backward, while 0x05 indicates motor left forward. Similarly, 0x06 indicates motor right backward and 0x07, motor right forward. (In fact, there is a naming error here: motor 2 actually signals the right motor, an error that was found later on and was corrected for on the client side due to time constraints).
4. The fourth byte indicates the motor speed. Since this is not a start byte, the MSB must be zero, and so the speed is indicated by the remaining seven bits, giving a range from 0 to 127.

The MotorHandler class also response to network status events. Namely, when nd is received, indicating a network disconnect, the motor handler will stop the motors.

7.2 Server Software

The server is a PHP socket server which accepts connections from rovers and from players. The rovers and the server exchange ASCII data, while the players and the server communicate using the WebSocket protocol which requires an initial handshake exchanging a security key, performed in ASCII, followed by exchanges of masked binary data. The server distinguishes plaintext TCP connections from WebSocket connections using the first message received: if the message starts with the connect prefix +, then the client is a rover TCP socket; if the message starts with GET, then the client is a player browser WebSocket.

The files for the server are found in `www/aricap/socket/`. The file `socket.php` is the main file that actually runs the server, while `Client.php` defines a class that abstracts the details of reading and writing to clients. The file `Websocket.php` contains the functions used to perform WebSocket handshakes and to mask and unmask WebSocket communications, while `config.php` defines the server parameters, such as the IP address and port.

We start by creating a socket, binding to an IP address and port, and listening on the socket. We also connect to a MySQL database to allow us to log all communications permanently for debugging. The socket server then loops indefinitely over the following steps:

1. From the list of socket resources, use `socket_select` to find the ones which have new data to read.
2. If the server socket is in the read list, then there is a new client. Accept the connection using `socket_accept` and create a new Client object.
3. For each of the clients in the read list, read the message using `Client::read`. The read function abstracts the protocol of the client:
 - a. If the client is using plain TCP, read uses `socket_read` to get ASCII data.
 - b. If the client is using WebSocket, read uses `socket_read` for the handshake and `socket_recv` after the handshake. The latter returns binary data which is then unmasked according to the rules of the WebSocket protocol, giving the original ASCII data.
4. The message is trimmed to ignore leading and trailing whitespace, and empty or whitespace-only messages are ignored.
5. The server checks to ensure that the client has declared its type and ID in a network connect message starting with `+`. Rovers send their details in their first message, while players send their details after the WebSocket handshake. If the details are valid, the server returns `:ok`; otherwise, it returns `:err`.
6. If a client resends a network connect message, the server checks if the details are the same as previously declared; if so, the network confirms the details by returning `:ok`; otherwise, it returns `:err`.
7. If the message starts with the network control character `:`, the server handles it the main loop:
 - a. `:d(etail)` is used by a client to request its details, which include the type, ID, and index of client among all the clients being served
 - b. `:q(uit)` is used by a client to close its connection
 - c. `:s(hutdown)` is used to shut down the server, closing all client connections
8. If the message does not start with a control character, then it is forwarded to `Client::handle`, which abstracts the type of the client:
 - a. If the client is a rover, `handle` forwards `ih` events to all players and returns `:ok`, while all other events return `:err`. The id of the source rover is appended to the event, such that an infrared hit originating from a client that connected with `+r2` is forwarded as `ih 2`.
 - b. If the client is a player, `handle` forwards the event to the rover with the matching ID. For example, a client that connected with `+p2` will have its message forwarded to the client that connected with `+r2`.
9. Messages are written to clients using `Client::write`, which abstracts the protocol of the client:
 - a. If the client is using plain TCP, then the ASCII string will be forwarded directly.
 - b. If the client is using WebSocket, then the ASCII string will be masked according to the rules of the WebSocket protocol, and the masked binary data will be forwarded to the client.

7.3 Player Software



The user interface (UI) is implemented in HTML5 using Scalable Vector Graphics (SVG) and WebSocket, and so should work on any modern browser, including IE 10+, Firefox 11+, Chrome 14+, Safari 6+, Opera 12.1+, is Safari 6+, Android Browser 4.4+, BlackBerry Browser 7+, Opera Mobile 12.1+, Chrome for Android 33+, Firefox for Android 26+, and IE mobile 10+ [1]. This allows for flexibility of use over a wide range of devices.

The files for the user interface are found in *www/arcap/game/*. The file *play.php* renders the overall page, which is styled using *play.css*. *Connection.js* defines a class that provides the methods for communicating with the socket server, while *DpadController.js*, *Dpad.js*, *DpadButton.js*, *SimpleButton.js*, and *StatusMarker.js* define the classes used to create the UI buttons and their corresponding callbacks.

The linking between the HTML and JavaScript elements of the page is performed using jQuery [2], while the UI itself is drawn using *svg.js* [10], a lightweight library for manipulating and animating SVG. The functionality is as follows:

- The direction pad (dpad) on the left is used to move the rover.
- The button on the right is used to shoot.
- The buttons in the middle indicate the status of the two rovers. They are green when the rovers are enabled, meaning they can move and shoot, flashes red when a rover has been shot, and then turn grey for six seconds, indicating that the rover is disabled and cannot move or shoot.
- When the players has been shot, the entire screen flashes red.

The url for the page shown above is *192.168.0.100/arcap/game/play.php?p=1*, which breaks down as:

- *192.168.0.100* is the IP address of our server

- `/arcap/game/play.php` is the path to our play page
- `?p=1` is a parameter selecting player 1
This is reflected in the header, which says *arcap one* for player 1 and *arcap two* for player two, and determines the network connect message that the browser will send to the socket server over WebSocket and thus which rover the player will control.

Using `svg.js`, we assign JavaScript callbacks to the *click* event of each of the buttons. Each callback uses the `Connection` class to send the appropriate message to the server. For example, the dpad up buttons sends the command `mbf100`, while the shoot button sends `is`.

The `WebSocket` object contained in the `Connection` class provides an `onmessage` callback which is used to respond to events from the server. The `Connection` object has a reference to its parent `controller`; upon receiving a hit event, the connection calls `controller.hit(...)` with the id of the source, which flashes the screen or status button as appropriate.

The red bar at the bottom of the screen indicates the connection status. When the `Connection` object receives an `:ok` message from the server, it calls `controller.ready(true)`, which turns the status bar green and allows the UI buttons to be pressed. When the `Connection` object receives an `:err` message from the server, it calls `controller.ready(false)`, which turns the status bar red and disables the UI buttons.

8 Test Plan

8.1 Software Test Plan

8.1.1 Command Interface

The command interface is a hardware-independent component that listens for network and infrared events and responds by sending messages up to the remote server or down to the infrared and motor hardware controllers. To test the command interface, mock objects are used to trigger test events and the response of the command handler is checked against the expected response.

- Test response to **infrared_receiver_update_task**
Confirm that command handler prepares an appropriate message and forwards it to the network handler successfully.
- Test response to **infrared_sender_update_task** of *shoot* command
Confirm that command handler parses command correctly and forwards to the infrared handler.
- Test response to **motor_handler_update_task** of *move* command
Confirm that command handler parses command correctly and forwards to the motor handler.

8.1.2 Network Interface

The network interface is a hardware-dependent component and so the following tests are integrated with the tests for the proper setup of the Xbee Wi-Fi module

- Test **TCP communications**
Confirm that rover is able to connect to remote server, send a request, and receive an OK response.

- Test **Wi-Fi response time**
Calculate delay in sending message to rovers from server. Calculate delay in sending messages from server to rover. Confirm that these delays are within acceptable limits.

These tests are hardware independent:

- Test behavior of **tcpSendAndConfirm**
Confirm that network handler can take a high-level description of messages to be send over the network and produce the corresponding HTTP request string or TCP payload.

8.1.3 Infrared Interface

The infrared interface is a hardware-dependent component and so the following tests are integrated with the tests for the proper setup of the infrared emitter and receiver diodes.

- Test **infrared emission**
Confirm that writes to memory-mapped locations for infrared emitter are reflected in the VHDL IR emitter component and on the GPIO pin to which the emitter will be connected.
- Test **infrared reception**
Confirm that reads from memory-mapped locations for infrared receptors reflect the state of the VHDL IR receiver component and the GPIO pins to which the receivers will be connected.
- Test **timer** or **GPIO interrupts**
Confirm that interrupts are triggered either periodically, according to a timer, or on the rising edges of the GPIO pins to which the receivers will be connected, depending on the architecture.
- Test **infrared receive queue** for overrun
Confirm that infrared handler responds to infrared events places in the infrared receive queue by the timer or GPIO interrupt ISR quickly enough to stop the quick from overrunning. Find the minimum time between consecutive infrared interrupts.

8.2 Hardware Test Plan

8.2.1 Infrared Emitter And Receiver

- Test **infrared emitter** with an infrared camera, camera without an IR filter, or other infrared sensitive device to confirm its signal. Also installed LED and programmed it to turn on during shoot event. Emitters were programmed to turn on when push button is pressed in the board.
- Test **infrared receiver** by connecting the receiver to an oscilloscope and using an infrared remote.
- Test **infrared emitter and receiver** together using an oscilloscope to confirm that the signal from the emitter is reflected at the receiver.
 - Test range of emitter and receiver to determine maximum shooting range.
 - Test response in various lighting conditions to determine impact of infrared noise from sunlight, incandescent bulbs, etc.
 - Test directionality of emitter and receiver.

8.2.2 Xbee Wi-Fi Module

- Modified its **destination IP address, SSID of target network it is connecting to, network type to infrastructure** and checked if the change was saved.
- Test **connection to Wi-Fi network** such as UWS. Confirm that the Wi-Fi module setup with WPA2 works as expected.
- Test **Internet Protocol** by pinging desktop over local network.
- Created a **talkback PHP server** and verified transmit and receive were working as expected.

8.2.3 Motor Controller

- Test **Motor Movement** by sending appropriate commands.
- Test **noise** caused by motors on power lines.
- Test desired motor move commands using LEDs in the motor controller.

Once all the components were working individually we integrated them and tested their functionality.

8.3 Integration Test Procedure:

- We integrated Wi-Fi and Infrared circuitry first and tested infrared emitter by sending command to shoot
- Infrared receivers were tested with appropriate software implementation to sending hit command to server
- Installed LEDs to test infrared emitter's functionality. The LED's were programmed to light up when infrared emitters went on.
- Tested if polling implementation was working as expected in software by sending commands and emitting infrared at receivers. Hit event was sent to server by rover
- Integrated motor controller to the system and tested by sending move commands to rover
- Tested if noise caused by motors were interfering with other signals using an oscilloscope
- To reduce noise capacitors were installed and tested noise caused by it in on reset and serial input lines of the motor controller.
- Noise was tested in pins for Wi-Fi module and capacitors were installed to reduce noise.
- LED was installed in the rover to display connection status of the Wi-Fi module. LED was programmed to turn on when connections were lost. Blinking LEDs represented trying to connect.
- Created a web application and tested move commands and shoot commands.
- Web application functionality of displaying hit event was tested.

9 Results and Characterization

9.1 Infrared Readings

The infrared receivers are phototransistors whose collector-emitter voltage increases with increasing infrared light. The analog signal from the emitters is read through the analog-to-digital converter on the DE0 Nano and read by the infrared handler as an integer value.

To test the behaviour of the receivers and emitters, we constructed a test circuit with one receiver beside three emitters. Because of this there is a bleed from the emitters to the receiver, which will be true for one of the four receivers on the rover. There is also ambient infrared light from the room lighting.

9.1.1 Ambient infrared readings

Context	Max Recorded Level
emitter off	2
emitters on	47

The strength of the signal recorded by the receivers is expected to fall off with the square of the distance. We measured the infrared readings using DVD remote with similar frequency and intensity as our three infrared emitters to characterize infrared reception versus distance.

9.1.2 Infrared readings vs distance

Distance (ft)	Max Recorded Level
0.5	246
1	243
1.5	235
2	180
2.5	131
3	118
4	86
5	70
6	65
7	57
8	55
9	54
10	51
11	51

10 Safety & Sustainability

(Unless Otherwise Stated, Calculations are Per Rover)

10.1 Wi-Fi Connection Failure

When a Wi-Fi connection failure occurs, the vehicle will stop moving.

10.2 Speed of Vehicle

Assuming we set the gear ratio to 58:1 (one of the two available options), use a conservative motor rpm value of 9000 (of the 9710 maximum), and estimate the radius of the bottom wheels to be 1.25cm (based on a total rover height of 2.125in), the speed of the vehicle is:

$$(9000/58) * (2 * \pi * 0.0125 / 60) = 20.31 \text{ cm/s}$$

A small rover at this velocity should not pose any significant threat to safety.

10.3 Voltage, Current, & Power

10.3.1 Motor Component

	Active Mode	Idle Mode
Max Current Draw	4.2 A (twice the stall current)	No current flowing and thus no power consumed.
Average Current Draw	1.5 A, driving on ECE 492 Lab Floor	
Voltage	3.638 V (measured 3 NiMH batteries)	
Max Power	4.2 A * 3.638 V = 15.28 W	
Mean Power	1.5 A * 3.638 V = 5.547 W	
Duty Cycle Proportions	90% in Active Mode 10% in Idle Mode; powering on/off, pausing to shoot, frozen	

10.3.2 Wi-Fi Component

	Transmit Mode	Receive Mode	AP Sleep Mode
Current Draw	260 mA	140 mA	1.2 mA
Max Voltage	3.64 V		
Mean Voltage	3.43 V		
Max Power (V * I)	3.64 V * 0.260 A = 0.9464 W		
Duty Cycle Proportion	5 %	40 %	55 %
Mean Power (V * I)	3.43 V * (0.05 * 0.260 A + 0.4 * 0.140 A + 0.55 * 0.0012 A) = 0.2389 W		

10.3.3 Infrared Circuitry

	3-Transmitter Circuit	1-Receiver Circuit
Max Voltage	1.6 V	Max Supply as Above; 3.64 V
Mean Voltage	1.2 V	Mean Supply as Above; 3.43 V
Operating Current	3 * 50 = 150 mA	2.08 mA – 3.60 mA
Max Power	1.6 V * 150 mA = 240 mW	3.64 V * 0.0036 A = 0.013104 W

Mean Power	1.2 V * 150 mA = 180 mW	3.43 V * 0.00284 A = 0.009741 W
Duty Cycle Proportions	On 5-10% of the time; Only for shooting	On 100% of the time; Polled for shot reception

10.3.4 Total Circuit Power

Max Power	Mean Power
15.28 W + 0.9464 W + 240 mW + 0.0524 mW = 16.5188 W	5.547 W + 0.2389 W + 180 mW + 0.0524 mW = 6.0183 W

10.4 Energy

10.4.1 Kinetic Energy

$$KE = \frac{1}{2} mv^2 = 0.5 \times 0.6 \text{ kg} \times (0.2031 \text{ m/s})^2 = \mathbf{0.012375 \text{ J}}$$

10.4.2 Stored/Potential Energy

4 AA NiMH Batteries for the On-Board Circuitry, and 3 AA NiMH Batteries for the Rover.

Batteries provided in the lab are rated at 2650 mAH, 1.2V.

$$\text{Energy} = 2650 * 1.2 * 7 / 1000 = \mathbf{22.26 \text{ Wh.}}$$

10.5 Summary of Safety Concerns

We do consider this project to merit any safety concerns. Power, voltage, speed, stored energy, and temperature will not reach dangerous levels.

10.6 Energy Cost

* Based on January 2014 energy price of 7.61¢ per kWh *

<http://www.directenergyregulatedservices.com/ELE/Electricity-Rates.aspx>

$$21.05 \text{ Wh} = 0.02105 \text{ KWh} * \$0.0761 = \$0.001670395 \text{ per hour, } * 24 * 365$$

$$= \mathbf{\$14.63 \text{ per year}}$$
 for continuous operation

10.7 CO₂ Production from an Equivalent Coal Generation of Energy

* Based on the 0.989 kg CO₂/KWh figure given in lecture *

$$21.05 \text{ Wh} = 0.02105 \text{ KWh} * 24 * 365 = 184.398 \text{ KWh/year} / 0.989 \text{ kg CO}_2 = \mathbf{186.449 \text{ kg CO}_2}$$

10.8 Solar Cell Area

A 1.2 KW solar package in the Edmonton area produces approximately 1500 KW/hour.

<http://www.efcl.org/EFCLActivities/SolarandEnergyProgram/tabid/449/Default.aspx>

The area of such a package can be found at 6' by 18' = 108 square feet.

<http://www.samsclub.com/sams/1-2kw-grid-tied-solar-power-system/prod1181196.ip>

$$\begin{aligned} \text{The area of solar cells required to power our project is thus is } & 0.02195 \text{ KW} / 1500 \text{ KW} \\ & = 0.0000146333 * 108 \text{ square feet} = 0.0015804 \text{ square feet} * (2.54 \text{ cm/in} * 12\text{in/ft})^2 \\ & = \mathbf{1.468 \text{ square centimeters}} \end{aligned}$$

11 Environmental Impact

11.1 Hazardous Substances/ RoHS Compliance

This project does not make use of any hazardous substances; it is RoHS compliant.

11.2 Other Environmental Points

The only aspect of this design that might be considered environmentally unfriendly is the use of batteries; this is mitigated by our use of rechargeable batteries.

12 References

- [1] Alexis Deveria. (2014). *Can I Use... Websockets* [Online]. Available: <http://caniuse.com/#feat=websockets>
- [2] The jQuery Foundation. (2014). *jQuery* [Online]. Available: <http://jquery.com/>
- [3] Michael Fischer. (2012). *Altera DE0 Nano* [Online]. Available: <http://www.emb4fun.com/fpga/nutos1/index.html>
- [4] Mozilla Developer Network .(2014, March 18). *WebSockets* [Online]. Available: <https://developer.mozilla.org/en/docs/WebSockets>
- [5] The PHP Group. (2001-2014). *Sockets* [Online]. Available: <http://www.php.net/manual/en/book.sockets.php>
- [6] Sann-Remy Chea. (2011, December). *Build a Real-Time Application Using HTML5 WebSockets* [Online]. Available: <http://srchea.com/build-a-real-time-application-using-html5-websockets>
- [7] Saran Chamling. (2013, May 4). *Simple Chat Using WebSocket and PHP Socket* [Online]. Available: <http://www.sanwebe.com/2013/05/chat-using-websocket-php-socket>
- [8] Terasic Technologies Inc. (2003-2013). *DE0-Nano Development and Education Board* [Online]. Available: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=1>
- [9] World Wide Web Consortium. (2012, September 20). *The WebSocket API* [Online]. Available: <http://www.w3.org/TR/websockets/>
- [10] Wout Fierens. (2012-2014). *svg.js* [Online]. Available: <http://www.svgjs.com/>

13 Appendix A Quick Start Manual

1. Statically assign the server an IP address. (Say, 192.168.0.101)
2. Install any web server stack such as WAMPP or XAMPP.
3. Copy contents of www folder to web folder of your server. (For example: www if using WAMPP or httpdocs if using XAMPP)
4. Start WAMPP and make sure server is running.
5. In WifiHandler.h modify the SSID to the Service Set ID of your network.
6. Change the SSID_KEY to the password of your network.
7. Change SERVER_ADDRESS to 192.168.0.101.
8. Change ROVER_ID in Messages.h to "r1" if programming Rover 1 or "r2" if programming Rover 2.
9. Save, clean and build the code.
10. Program ARCAP.elf and de0_nano_system.sof file into FPGA's non-volatile memory.
11. In the terminal, go to the directory www/arcap/socket and execute "php socket.php"
12. Turn on both the battery power packs.
13. Wait until the red LED stops blinking. (Blinking represents trying to connect, if does not connect check socket.php is running and all verify all network ID's are correct)
14. Open web browser and connect to "192.168.0.101". The following webpage is displayed and select the Rover you want to control.



15. The following is the control interface for the rover. The right button represents shoot. The left button is to control the direction of the rover.



14 Appendix B Future Features:

14.1 Future Feature 1:

The first feature to add would be to encode the infrared signals in order to achieve greater range. Encoding the signals will also allow us to have different types of shots that will have different effects. For example; one type of shot could freeze the opponent, and one type of shot could cause the opponent to drop the flag.

14.2 Future Feature 2:

Another goal is to transition from a web app to an Android app. This would allow for players to operate more flexibly, and in a larger game space, moving around with the cars that they're controlling.

14.3 Future Feature 3:

After the above additions, we would want to add a real-time video feed of the car's view and perhaps of the game map as well, so that remote gameplay can be achieved. We would likely do this by interfacing a camera through USB.

14.3.1 Details of Operation with GPIO Camera

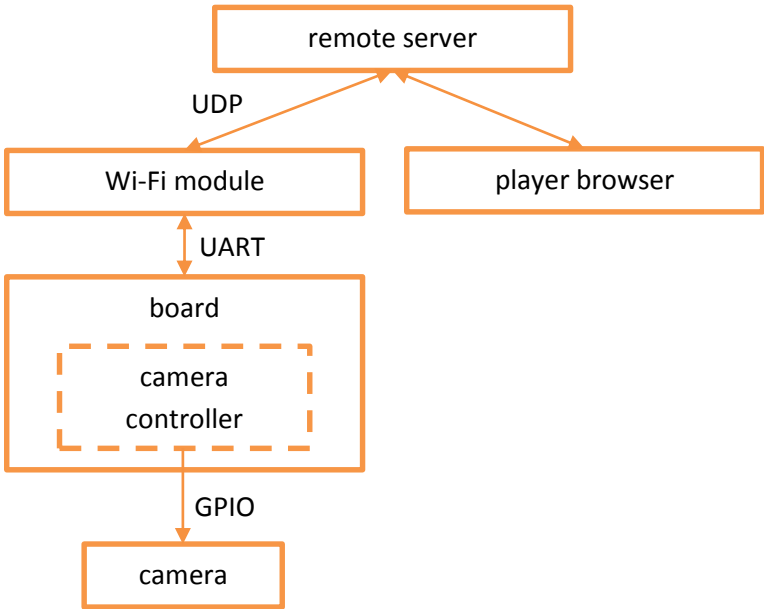


Figure 1 Integration of network communications with GPIO camera

Terrasic provides a 5 MP Digital Camera Development kit, the TRDB D5M, which interfaces to several Alterra boards, including the DE2 and DE0 Nano, over GPIO. This camera provides a maximum data rate of 96 Mp/s at a master clock rate of 96 MHz with a programmable frame rate of 15 fps at maximum resolution increasing to 70 fps when the resolution is reduced to VGA (640 x 480). These specifications suggest that transfer of data from the camera to the board may approach rates feasible for our application, particularly as resolution is reduced, but transfer between the board and the Wi-Fi module,

and then between the Wi-Fi module and the remote server, creates another bottleneck since the RN 171 EK Wi-Fi module claims data rates up to 500 Kbps over UART. In this setup, the live camera feed is streamed, over GPIO, to the board, and then, over UART, to the Wi-Fi module, where it is transmitted using either TCP or UDP to the remote server (UDP trades an increased probability of data loss for throughput). Image processing is applied at the server to identify targets, using either preexisting libraries for QR code recognition or OpenCV for more general object recognition, and positive results are recorded and pushed to the players.

14.3.2 Details of Operation with IP Camera

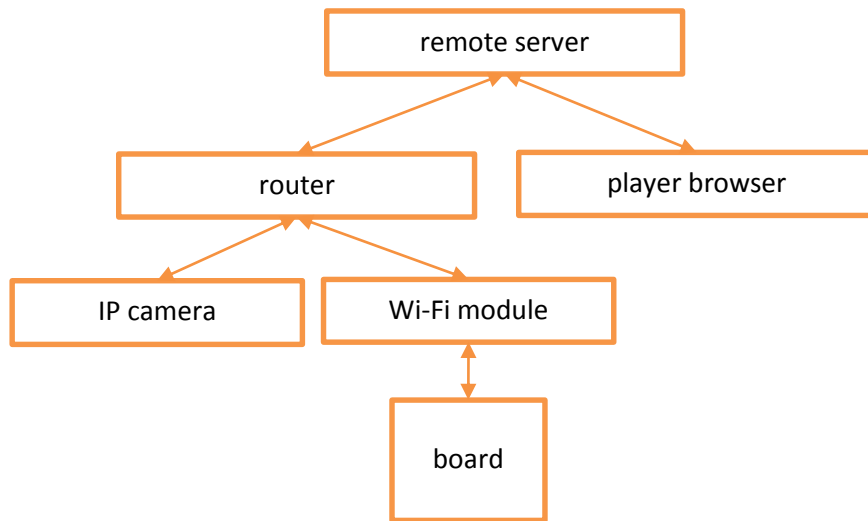


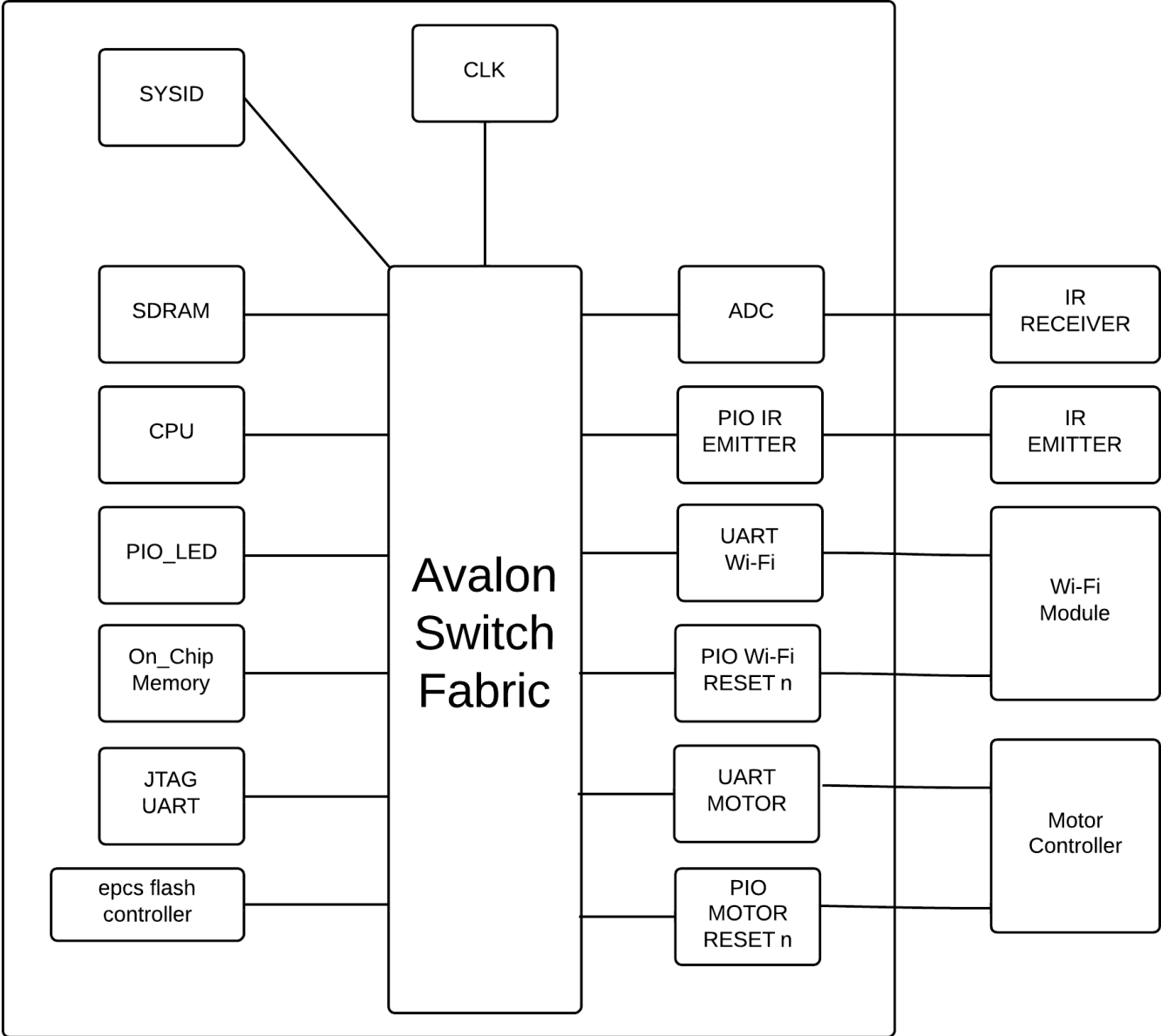
Figure 2 Integration of network communications with IP camera

As an alternative to routing the camera feed through the microcontroller, which introduces several bottlenecks, one potential design uses a separately networked recording device such as an internet protocol (IP) camera. IP cameras, which widely used in security applications, come prepackaged with the ability to stream over Ethernet or Wi-Fi and to view the stream remotely. IP cameras typically need to be paired with a dedicated router and come in varying ranges in terms of their maximum distance from that router, where increasing range comes with increasing price. IP cameras are often linked to proprietary end-user software, which is of limited use for our project; however, with appropriate use of port forwarding, it may be possible to access the raw camera feed and extract frames for image processing.

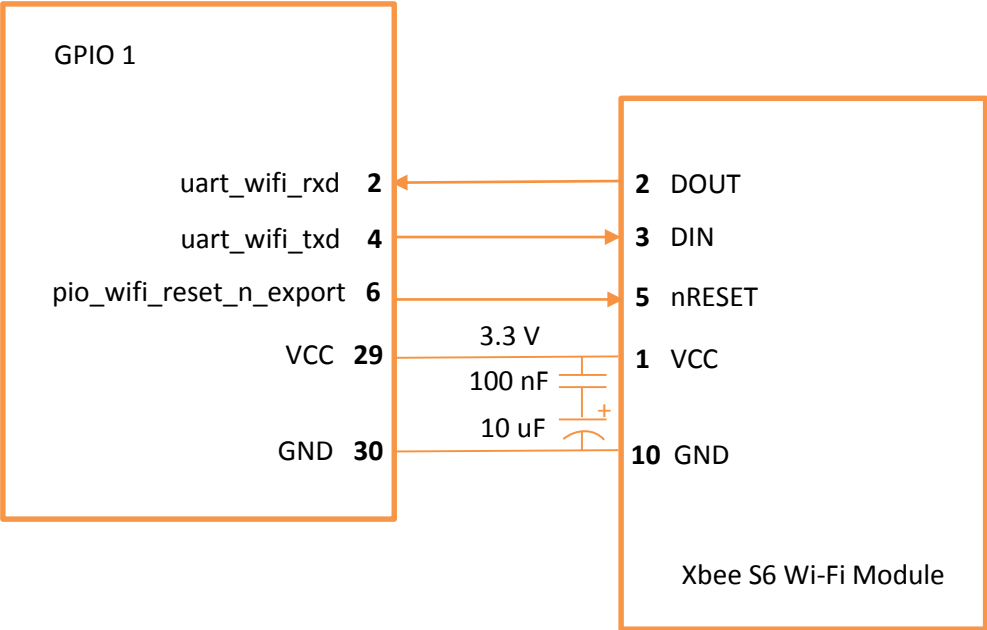
In a design using an IP camera, the board maintains a network connection solely for commands; as in the infrared case, this results in a significantly smaller data throughput requirements for our microcontroller which is more appropriate to its resources.

15 Appendix C Hardware Documentation

15.1 Top-Level

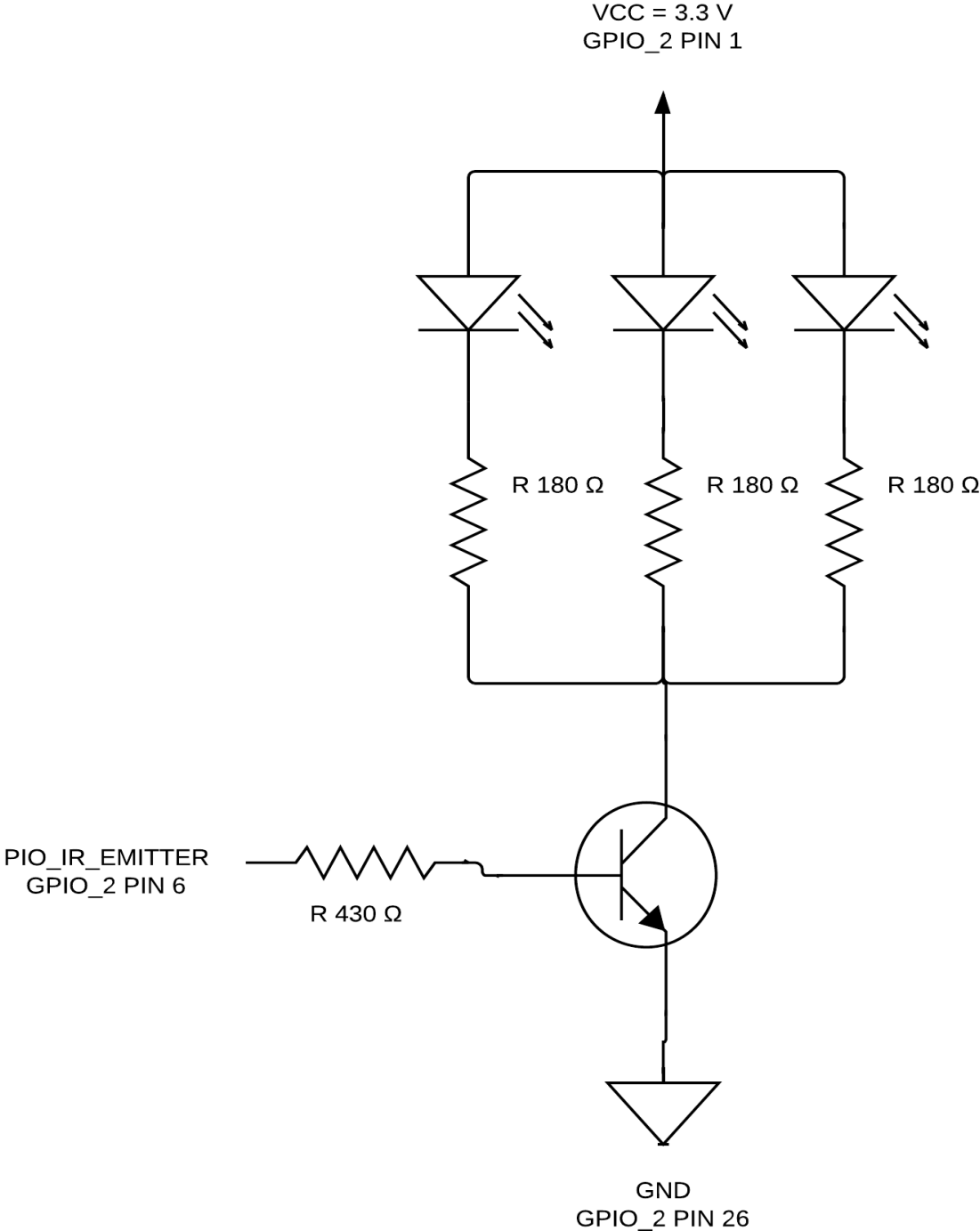


15.2 Wi-Fi

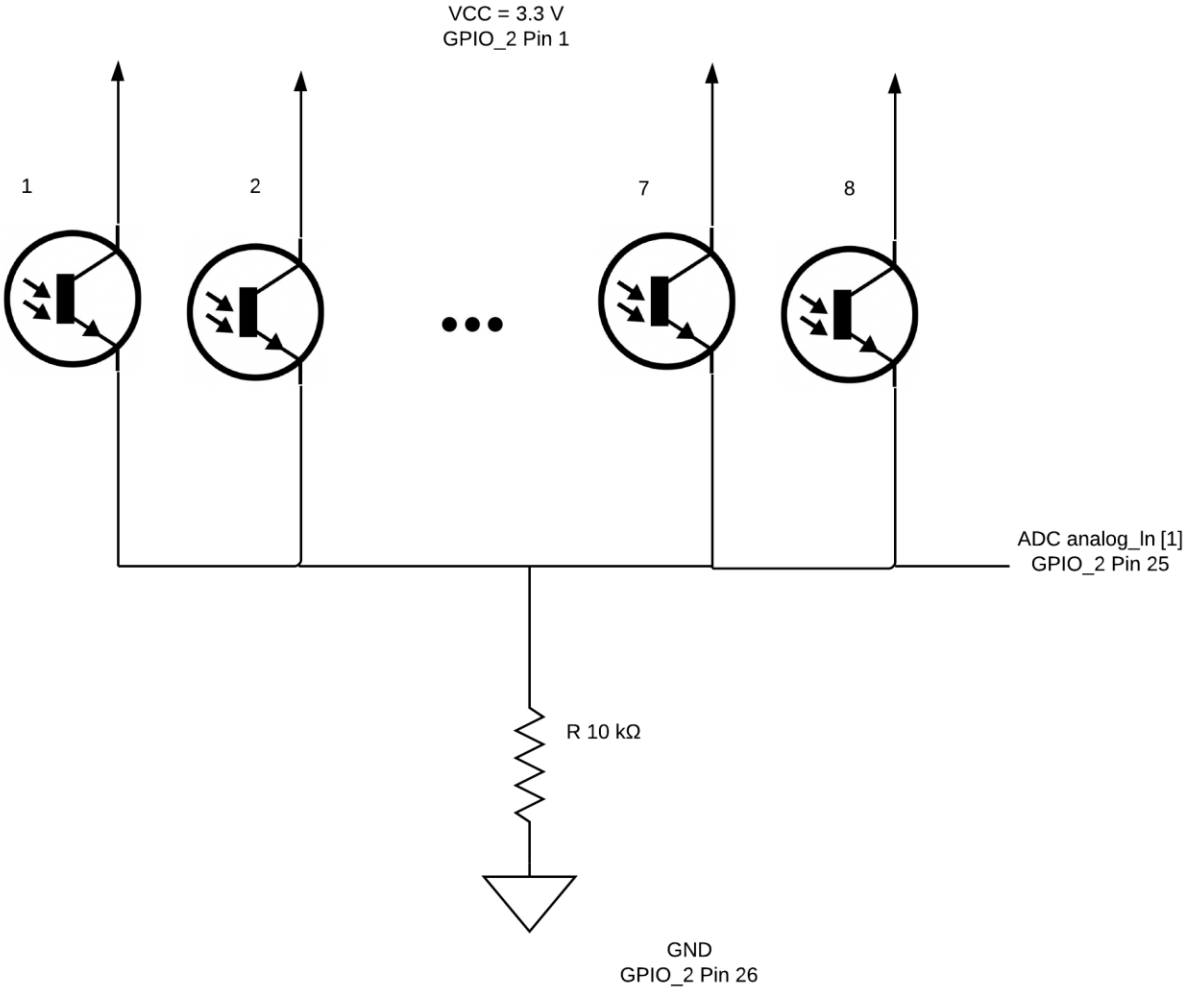


15.3 Infrared

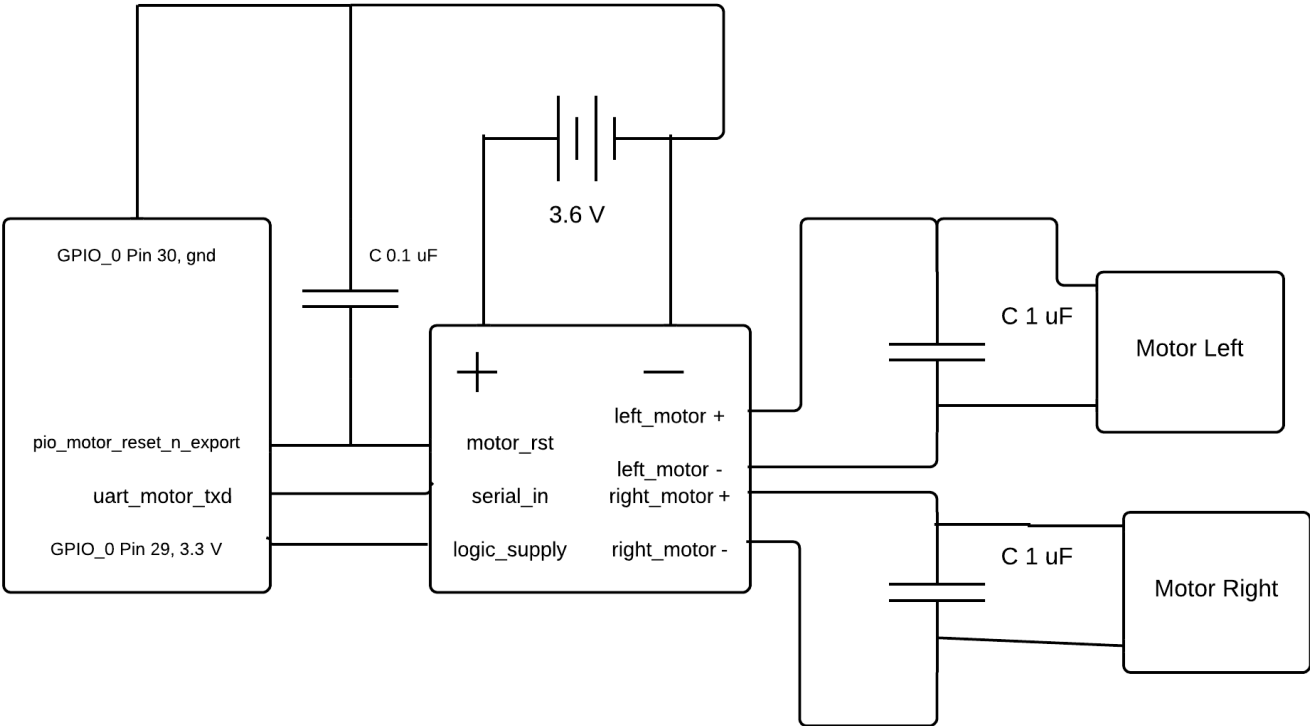
15.3.1 Emitters



15.3.2 Receivers



15.3.3 Motor



16 Appendix D Source Code

16.1 Index to Source Code

Module	Description	Status (T = Tested & Passed)
InfraredReceiver.cpp InfraredReceiver.h	Reads input from the infrared receivers through the analog-to-digital converter (ADC) and posts infrared hit events to a listener.	T
InfraredSender.cpp InfraredSender.h	Listens for send commands on a queue and applies them to the infrared emitters.	T
Listener.cpp Listener.h	The bridge abstract class provides a common structure for classes that listen to data sources and forward relevant events to recipients.	T
MotorHandler.cpp MotorHandler.h	Listens for commands on a queue and uses them to control the motor. The command must specify: (a) which motor to run <code>{l(ef), r(ight)}</code> , (b) the direction to run the motor in <code>{{(f)orward, (b)ackward}</code> , (c) the speed to run the motor at <code>[0 to 127]</code> .	T
NetworkReceiver.cpp NetworkReceiver.h	Listens for messages from the server using a wifi handler and forwards them to handlers that execute the corresponding effects on the rover.	T
NetworkSender.cpp NetworkSender.h	Listens for events from handlers on the rover and forwards them to the server using a wifi handler.	T
Status.h	Contains status info for the project, such as classes of exceptions that may occur.	T
Tasks.h	Contains definitions of tasks, stack sizes and priorities.	T
WifiHandler.cpp WifiHandler.h	The wifi handler provides a high-level view of the interfacing to the Xbee wifi module.	T

16.2 Data Flow

