**EE175 Final Report**

**SunE-Bot**

**EE 175 Final Report**

**Department of Electrical Engineering, UC Riverside**

|  |  |
| --- | --- |
| Project Team Member(s) | Christopher Verdegan, James Thi, Yidi Wang |
| Date Submitted | March 16, 2018 |
| Section  Professor | Konstantinos Karydis |
| Revision | 2.0 |
| URL of Project Wiki/Webpage | Google Drive: https ://drive.google.com/drive/folders/0Byn20I8b1E-sZF9DRGFsUnV0Vm8  Github: https://github.com/cverd001/Senior-Design |
| Permanent Emails of all team members | [verdeganc@gmail.com](mailto:verdeganc@gmail.com), JamestheThi@gmail.com, ydwang21@gmail.com |

Summary

This report presents the specification of the SunE-Bot Senior Design project completed during the 2017-2018 school year at University of California, Riverside.

Revisions

| Version | Description of Version | Author(s) | Date Completed | Approval |
| --- | --- | --- | --- | --- |
| 1.0 | First draft, some parts still need completion and reformatting | Christopher Verdegan, James Thi, Yidi Wang | 03/03/2018 |  |
| 2.0 | Final Draft, finished missing sections and made correction based off of Professor Karydis’s feedback | Christopher Verdegan, James Thi, Yidi Wang | 03/16/2018 |  |

Table of Contents

[Revisions 2](#_Toc509004866)

[Table of Contents 3](#_Toc509004867)

[1 Executive Summary 6](#_Toc509004868)

[2 Introduction 7](#_Toc509004869)

[2.1 Design Objectives and System Overview 7](#_Toc509004870)

[2.2 Backgrounds and Prior Art 8](#_Toc509004871)

[2.3 Development Environment and Tools 8](#_Toc509004872)

[2.4 Related Documents and Supporting Materials 9](#_Toc509004873)

[2.5 Definitions and Acronyms 10](#_Toc509004874)

[3 Design Considerations 11](#_Toc509004875)

[3.1 Realistic Constraints 11](#_Toc509004876)

[3.2 System Environment and External Interfaces 12](#_Toc509004877)

[3.3 Industry Standards 12](#_Toc509004878)

[3.4 Knowledge and Skills 12](#_Toc509004879)

[3.5 Budget and Cost Analysis 14](#_Toc509004880)

[3.6 Safety 15](#_Toc509004881)

[3.7 Performance, Security, Quality, Reliability, Aesthetics etc. 15](#_Toc509004882)

[3.8 Documentation 16](#_Toc509004883)

[3.9 Risks and Volatile Areas 16](#_Toc509004884)

[4 Experiment Design and Feasibility Study 17](#_Toc509004885)

[4.1 Experiment Design 17](#_Toc509004886)

[4.2 Experiment Results, Data Analysis and Feasibility 20](#_Toc509004887)

[5 Architecture and High Level Design 23](#_Toc509004888)

[5.1 System Architecture and Design 23](#_Toc509004889)

[5.2 Hardware Architecture 23](#_Toc509004890)

[5.3 Software Architecture 24](#_Toc509004891)

[5.4 Rationale and Alternatives 26](#_Toc509004892)

[6 Low Level Design 28](#_Toc509004893)

[6.1 Module 1: Teensy 3.6 Module 28](#_Toc509004894)

[6.1.1 Processing narrative for Teensy 3.6 28](#_Toc509004895)

[6.1.2 Teensy 3.6 Interface Description 29](#_Toc509004896)

[6.1.3 Teensy 3.6 Processing Details 29](#_Toc509004897)

[6.2 Module 2: Charging Circuit Module 29](#_Toc509004901)

[6.2.1 Processing narrative for Charging Circuit 29](#_Toc509004902)

[6.2.2 Charging Circuit Interface Description 30](#_Toc509004903)

[6.2.3 Charging Circuit Processing Details 30](#_Toc509004904)

[6.3 Module 3: Movement Module 30](#_Toc509004905)

[6.3.1 Processing narrative for Movement 30](#_Toc509004906)

[6.3.2 Movement Interface Description 31](#_Toc509004907)

[6.3.3 Movement Processing Details 31](#_Toc509004908)

[6.4 Module 4: Light Sensor Module 31](#_Toc509004909)

[6.4.1 Processing narrative for Light Sensor 31](#_Toc509004910)

[6.4.2 Light Sensor Interface Description 31](#_Toc509004911)

[6.4.3 Light Sensor Processing Details 32](#_Toc509004912)

[6.5 Module 5: IMU Module 32](#_Toc509004913)

[6.5.1 Processing narrative for IMU 32](#_Toc509004914)

[6.5.2 IMU Interface Description 32](#_Toc509004915)

[6.5.3 IMU Processing Details 33](#_Toc509004916)

[6.6 Module 6: Encoders Module 33](#_Toc509004917)

[6.6.1 Processing narrative for Encoders 33](#_Toc509004918)

[6.6.2 Encoders Interface Description 33](#_Toc509004919)

[6.6.3 Encoders Processing Details 33](#_Toc509004920)

[6.7 Module 6: Raspberry Pi and Picamera Module 34](#_Toc509004921)

[6.7.1 Raspberry Pi and Picamera Processing Details 34](#_Toc509004922)

[6.7.2 Raspberry Pi and Picamera Interface Description 34](#_Toc509004923)

[6.7.3 Raspberry Pi and Picamera Processing Details 35](#_Toc509004924)

[7 Technical Problem Solving 36](#_Toc509004925)

[7.1 The Power Buses Problem 36](#_Toc509004926)

[7.2 Solving the Power Buses Problem 36](#_Toc509004927)

[7.3 The Charging Problem 36](#_Toc509004928)

[7.4 Solving the Charging Problem 36](#_Toc509004929)

[7.5 The Encoder Problem 36](#_Toc509004930)

[7.6 Solving Encoder Problem 37](#_Toc509004931)

[7.7 The Turning Problem 37](#_Toc509004932)

[7.8 Solving the Turning Problem 37](#_Toc509004933)

[8 User Interface Design 38](#_Toc509004934)

[8.1 Application Control 38](#_Toc509004935)

[8.2 User Interface Screens 38](#_Toc509004936)

[9 Test Plan 39](#_Toc509004937)

[9.1 Test 1: Computer Vision Bright Spot Detection 39](#_Toc509004938)

[9.2 Test 2: Communication between the RPI and the Teensy 39](#_Toc509004939)

[9.3 Test 3: Spinning and Tracking 39](#_Toc509004940)

[9.4 Test 4: GSA 40](#_Toc509004941)

[9.5 Test 5: Return to Max 40](#_Toc509004942)

[9.6 Bug Tracking 40](#_Toc509004943)

[9.7 Quality Control 41](#_Toc509004944)

[9.8 Identification of critical components 42](#_Toc509004945)

[9.9 Items Not Tested by the Experiments 42](#_Toc509004946)

[10 Test Report 44](#_Toc509004947)

[10.1 Test 1: Computer Vision Bright Spot Detection 44](#_Toc509004948)

[10.2 Test 2: Communication between the RPI and the Teensy 44](#_Toc509004949)

[10.3 Test 3: Spinning and Tracking 44](#_Toc509004950)

[10.4 Test 4: GSA 45](#_Toc509004951)

[10.5 Test 5: Return to Max 45](#_Toc509004952)

[11 Conclusion and Future Work 46](#_Toc509004953)

[11.1 Conclusion 46](#_Toc509004954)

[11.2 Future Work 47](#_Toc509004955)

[11.3 Acknowledgement 48](#_Toc509004956)

[12 References 49](#_Toc509004957)

[13 Appendices 50](#_Toc509004958)

# Executive Summary

Our goal was to create a robot with the functionality of finding the best lit spot in a certain area to charge its battery using a solar panel. As technology continues to improve and costs go down, autonomous robots are becoming increasingly prevalent, and generating their own energy allows for even more autonomy. Autonomous robots can be left out in the field to collect data, and by being self-sustaining they can save resources and time in efforts of recharging. Harnessing solar power is the perfect option, because it is a clean source of energy that requires little to no maintenance. However, solar panels are very reliant on being positioned in a well-lit area facing towards the sun for maximum power generation. Today, we have technologies such as security robots operating in cities and data gathering rovers operating on Mars. With the proliferation of robotics, we will soon see more of these autonomous robots operating in more places. Wherever there are trees, tall buildings, or any other tall shadow casting structures, autonomous robots can waste valuable time charging in shaded areas. Any sort of autonomous robot with mobility should have the capability of repositioning itself in the best lit spot. We wanted to create the best method to find the optimal charging spot so that the robot can charge as fast as possible. To do this, we created two search methods. One which uses sensor readings while traversing a size-adjustable grid to determine its ideal location and one by finding the brightest spot through computer vision. After finding and positioning itself in this spot, we programmed the robot to rotate itself so that the solar panel would directly face towards the sun, and to have a functionality to continuously adjust its position to track the sun to maximize its power generation. We made the grid-size adjustable to the user’s preference and created a menu so that grid-size and computer vision algorithm can be chosen without having to connect the SunE-Bot to a computer and modify the software. With the capability of implementing any sort of sensor or actuator, the SunE-Bot provides countless opportunities for autonomous robotic missions.

# Introduction

## Design Objectives and System Overview

This is a design for a basic autonomous robot capable of repositioning itself to an optimal sun-lit location to self-sustain off solar power. This is meant to be a basic functionality – an ideal “charging mode” function which any mobile solar-powered robot can implement. This functionality is geared towards rover-type robots which have a means of mobility and operate without the presence of humans. Robots with our self-sustaining design are free to explore and gather data on their own without any assistance. By developing the basic infrastructure of light-search algorithm and solar-charging circuitry, our design can be applied to countless autonomous robots that exist and will be developed in the future.

This design addresses current issues in renewable resource usage and in manpower. Development of this design implements and promotes a clean and reusable energy resource while also leaving the taxing, expensive tasks of data-gathering to the robots.

Most of the algorithms and computation are performed on a Teensy 3.6 microcontroller (Arduino variant) which sends signals out to all other peripherals. Light sensors in the form of photoresistors (4) and a solar panel send input to the Teensy, which uses the data in its algorithm to find an optimal bright location. The Teensy controls 2 motor drivers, which each control 2 motors to move the entire mobile car. Other sensors involved in repositioning include motor encoders, an Inertial Measurement Unit (IMU), and an action camera attached to a Raspberry Pi microcontroller. The solar panel is also attached to a charge controller and voltage regulator to constantly charge a battery under enough light. Configuration of the Teensy and Raspberry Pi are done in C and Python, respectively.

The robot contains two different algorithms to detect light. The first is a Grid Search Algorithm (GSA). The robot traverses a set grid and periodically reads voltages through its sensors. If the threshold for complete sunlight is met, the robot stops and marks that as the optimal location. If the entire grid has been travelled through without hitting the threshold, the robot swivels and returns to the spot that returned the highest light reading. The other algorithm is the Computer Vision Algorithm (CVA). This method has the robot spin in place and uses the camera to check for any bright spots before moving in the direction with the brightest spot. This method does not usually require the motors to be in use for as long but requires more voltage to power the Raspberry Pi which also powers the camera.

At start up, the user has the option of selecting the algorithm to use, as well as how big of a grid to set if the GSA is selected. This is done through turning of the back-left wheel and covering of the front photoresistors.

Technical Design Objectives:

* The robot should be able to find the optimal bright location in a reasonable time frame. The energy expended to reposition itself should be less than energy stored during 1 hour of charge time. This holds true for both the GSA and the CVA.
* The full charge time of the batteries should be less than 6 hours. It is important that the batteries charge fully during the daylight, while there is still sunlight. This allows the robot to charge during the day and operate at night if needed.
* The robot should be able to find the optimal bright location in a 10ft x 10ft area at a 95% accuracy and stop within a 15cm radius of the light spot. This applies to both algorithms.

Responsibilities:

* Chris headed the construction of power system circuitry. This circuitry connects the solar panel to the battery as well as its input to the Teensy. Chris also handled the many iterations of the measured movement methods (delay, encoder, compass IMU). He also developed the menu functionality and indication sounds. He outlined the GSA and developed the receiving end of the CVA. He also designed the 3D printed parts used to position the light sensors and the solar panel.
* Yidi headed the assembly of the robot chassis and tested basic robot motion. She configured the motor drivers and ensured communication was correct between the Teensy, the motor drivers and the motors. In addition, Yidi worked with the camera and Raspberry Pi, and developed the image processing and the function to detect light spots using the Pi.
* James lead the development of the Grid Search Algorithm. It is a computational heavy algorithm to account for various grid sizes and various angle calculations to arrive at the correct bright spot. James worked on the algorithms to ensure it was as data and time efficient as possible. In addition, James contributed to parts in power system circuitry and robot precise turning.

## Backgrounds and Prior Art

While conducting research on possible design ideas we came across a “Sun-Seeking Robot” online. This little mobile robot holds a planter on its back and roams indoors for the sunniest spot by randomly moving until it hits sunlight. It detects sunshine with the help of two solar panels, and to have it differentiate between indoor lighting and true sunlight. It is a small project designed to provide a living plant with ample sunlight without the need for constant human attention.

This design was a small inspiration to the design that we came up with. While this existing robot roams in the house to care for a plant, we decided to repurpose it for outdoor activity. With exploratory rovers being more common now – even on other planets – we decided to develop something similar in the form of a self-sustaining autonomous robot. We also wanted to add more intelligence to the light searching method, instead of simply random movement.

## Development Environment and Tools

The bulk of this project was done in the Arduino integrated development environment with C++. Arduino software was selected for its many open-source libraries that provide support for robotics. This works well with the chosen microcontroller, Teensy 3.6, because it is compatible with the Arduino IDE. The IDE and microcontroller were used to develop most of the function of the robot, outside from the image analysis done on CVA.

In conjunction with the Teensy 3.6, a Raspberry Pi 3 compact computer is used for the image processing side of the CVA. It is run on the Raspbian operating system in the Python language. Analysis of the image feed was performed with OpenCV (Open Source Computer Vision Library), a library of real-time computer vision functions. The camera is a Picamera 1.13 and is accessed through Picamera Python library. RPI.GPIO python library is used to access the GPIO pins.

For file sharing we used Git, which also provided us a version control system for our software.

CAD modeling of the 3D printed parts was done using the SolidWorks 2016 software.

For hardware testing, we used the Hewlett Packard E3630A DC power supply, the Tektronix TDS  340 oscilloscope, and the Fluke 45 multimeter. We also used various handheld multimeters for outdoor testing of the sensors.

## Related Documents and Supporting Materials

**Teensy 3.6 Microcontroller:** This microcontroller is the key computer of the current prototype. The Teensy communicates with all peripherals on the robot and runs both light-searching algorithms.

<https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/K66P144M180SF5RMV2.pdf>

**Raspberry Pi 3:** This computer communicates with the camera attached to the robot, processes the data given and sends data back to the Teensy regarding the CVA.

<https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf>

**Charge Controller-TP4056**: We used this component to connect the solar panel to the battery to remove the risk of overcharging.

<https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf>

**L298N Dual H Bridge:** This component is the motor driver used to bridge the connection between the Teensy microcontroller and the motors. We used two of these drivers to modify the speed of the motors.

<https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf>

**IMU-MPU6050:** This component was used for the robot to determine its angular position. This was key to making accurate turns.

<https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf>

**OpenCV:** Also known as Open Source Computer Vision Library, this set of computer vision libraries and functions was used as support when programming the CVA.

**Arduino IDE:** This integrated development environment was used to program all functions and algorithms on the Teensy 3.6 microcontroller**.**

**I2C:** This communication standard was used to communicate between the IMU and the Teensy 3.6.

**IEEE 802.11:** This WiFi communication standard was used to communicate remotely between the RPI (Raspberry Pi) and SSH (Secure Shell).

**USB:** This communication standard was used to communicate to and program the Teensy 3.6 from a personal computer.

## Definitions and Acronyms

**Common Terms**

**ADC**- Analog to digital conversion, used when reading data from our peripherals.

**GPIO** - General Purpose Input/Output (pins): Generic pins used to connect the microcontrollers to peripherals.

**Li-on** - Lithium-ion: A type of rechargeable battery used in the SunE-Bot.

**18650** - 18mm by 65mm battery, used in the SunE-Bot.

**PWM** - Pulse Width Modulation: Technique for getting analog results with digital means. Used in SunE-Bot’s motors.

**Tick** - One high-low output reading of a digital signal. Used to count readings emitted by motor encoders.

**IMU** - Inertial Measurement Unit: An electronic device to measure the direction of the SunE-Bot.

**CAD** - Computer-aided design.

**RPI** - Raspberry Pi 3 Model B: A small computer to control the attached camera.

**Project-Specific Terms**

**GSA** - Grid Search Algorithm: SunE-Bot’s first algorithm - travels through a grid of set size and searches for an optimal bright spot.

**RTM** - Return to Max: A key function in the GSA - after searching the entire grid, head back to the brightest spot detected.

**CVA** - Computer Search Algorithm: SunE-Bot’s second algorithm - uses computer vision to search around itself for an optimal bright spot before heading toward it.

# Design Considerations

## Realistic Constraints

**User Constraints**

We created a simple menu for the user to interact with the robot upon start up. This allows the user to select the algorithm for the robot to perform, as well as the grid size for the GSA. This menu, however, is communicated to the user through a series of buzzer noises which may not be intuitive enough. A user manual or knowledge of the menu’s audio interface is required before using the robot.

**Power Consumption Constraints**

One of the goals when developing this project was to have the robot perform the algorithm as quickly as possible and as energy-efficiently as possible. We experimented with meeting those goals with two separate algorithms. The Computer Vision algorithm uses much less power to the motors, but with the trade-off of increased power to the Raspberry Pi. Under good weather, the GSA should be more energy efficient. Also if there are few light spots through a large area to scan, the CVA would save more time than the GSA.

**Voltage/Current Supply Constraints**

Under good sunlight conditions the solar panel emits up to 14V and up to 400mA when connected to the load. However, the 18650 batteries cannot take that much voltage/current without overcharging. We attached a voltage regulator and charge controller between the two to ensure safe charging. This prevents reverse current as well.

Regarding powering the robot, the motor drivers need at least 7 volts to power the motors and function properly. In comparison the Teensy microcontroller can only accept a maximum of 6.0 volts as power. To fix this issue we attached two 18650 batteries serially to power the motor. Only one of the batteries in series is used to power the Teensy.

**Weight/Size Constraints**

Because this robot is meant to be mobile and on its own, we designed it with weight and size in mind. We kept the robot as light as possible for more energy efficiency, and as small as possible to allow it to maneuver in any tight spaces if necessary.

## System Environment and External Interfaces

The main processor, the Teensy 3.6 microcontroller, is programmed in the Arduino IDE in the C language. The second processor is a Raspberry Pi 3.0 controller used to control the camera and is programmed in Python. The two microcontrollers communicate through two high/low bits to signal when to start the camera and when a light source is found. However, when programmed, the robot does not require any further interaction with the code, the IDE or bits by the user. The robot operates through on/off switches and wheel-turning as controls.

## Industry Standards

**IEEE 802.11** - this is a set of MAC and PHY specifications for implementing wireless local area network (WLAN) communication. This standard is used to communicate remotely with the RPI controller through SSH (secure shell).

**I2C** - this connection is used to attach lower-speed peripherals/sensors to processors and microcontrollers through a multi-master, multi-slave system. This standard is used for communication between the IMU sensor and the Teensy microcontroller.

**USB (Universal Serial Bus)** - is an industry standard to define connections from peripheral devices to personal computers. It is a standardized connection for personal computers to communicate and supply power to peripherals such as a keyboard, computer mouse, or in this case, the Teensy 3.6.

## Knowledge and Skills

**James Thi**

Prior Coursework:

* CS10: Intro to Computer Science
* CS13: Computer Science for Electrical Engineering Majors
* CS14: Introduction to Data Structures and Algorithms
* CS141: Intermediate Data Structures and Algorithms
* EE1A/B: Engineering Circuit Analysis
* EE100A: Electronic Circuits
* EE120A/B: Logic Design/Intro to Embedded Systems

Knowledge/Skills Prior to Project:

I have had experience with basic embedded systems from CS120b and CS122a, as well as coding in C from my previous CS classes.

Knowledge/Skills Learned from Project:

Throughout this project I learned an immense amount in the areas of power charging circuitry, in learning the basics of a new microcontroller (Teensy), in clock timing and developing algorithms. The coding experience I had before helped me a lot and this project refreshed my skills and improved them by a lot. I learned to work in the Arduino IDE, learned to push my code to GitHub to handle version control, and learned to use the code to handle and control peripheral sensors and motors. I learned how to use the various peripherals in the project including a compass, an IMU, motor encoders, a solar panel, and how to look at and study the datasheets for each. I also learned how to better collaborate in a team and manage the timing of a big long-term project.

**Christopher Verdegan**

Prior Coursework:

* CS10: Intro to Computer Science
* CS13: Computer Science for Electrical Engineering Majors
* EE1A/B: Engineering Circuit Analysis
* EE100A/B: Electronic Circuits
* EE120A/B: Logic Design/Intro to Embedded Systems
* EE123: Power Electronics

Knowledge/Skills Prior to Project:

I have had experience working with autonomous robotics from competing in the IEEE hosted Micromouse competitions. This taught me the basics of controlling motors using encoders and how to program sensor input based decisions. I had also worked with a different version of the Teensy microcontroller we used. From this project, I also learned how to properly build and wire robotic parts, as well as how to solder.

Knowledge/Skills Learned from Project:

Though I had worked with autonomous robotics in the past, I had a lot more help with peer mentors and preset guidelines so this project back then. With SunE-Bot, I had to relearn a lot of the basics of robotics and ensure that I understood it because I no longer had the support of the peer mentors to depend on. I learned a lot about debugging problems between hardware and software. I learned to use things I had studied in electronic circuit classes for debugging. With debugging of the optical encoders, I used a digital oscilloscope to read its outputs. Though I had learned the use of oscilloscopes in my EE100A course, discovering the use of it in a practical application was new to me. I also learned how to do 3D CAD with SolidWorks during this project. Since the SunE-Bot needed six parts to hold up the sensors and solar panel, I taught myself the basics of SolidWorks and made custom parts that fit directly into the cutouts on the plastic of the robot’s body. I also learned about interfacing with new components such as the IMU, digital compass, solar panel, and optical encoders which I had never worked with before.

**Yidi Wang**

Prior Coursework:

* EE128: Data Acquisition, Instrumentation and Process Control
* EE146: Computer Vision

Knowledge/Skills Prior to Project

I had learned implement C++ to control a robot from the RoboCup China Open competition before in my home university. The experience taught me how to code in C++, interact with the robot on Linux OS and draw circuit schematics. From EE128, I learned basic control of motors using motor drivers and the principles of I2C communication. Also, from EE146, I learned basic theory of technical methods of image processing.

Knowledge/Skills Learned from Project

The knowledge and skills I came in with were not adequate for the Senior Design project. During the project, I recalled many concepts that I had almost forgot, for example, the hardware implementation of a low-pass filter, which was then used for the encoders. Additionally, in order to implement computer vision on the RPI, I learned Python OpenCV and the usage of the RPI and the camera by myself.

## Budget and Cost Analysis

Table of the retail costs of all parts used in the final iteration of SunE-Bot. If purchased in bulk directly from the supplier, these prices would be significantly lower. Parts such as wires, resistors, buzzer, capacitors, 3D printed holders, through hole board are not listed as their individual costs are all negligible (<$1).

|  |  |  |
| --- | --- | --- |
| **Part** | **Retail Cost [$]** | **Vendor** |
| **Raspberry Pi 3 (including heating sink and charging cable)** | 48.50 | AliExpress |
| **Teensy 3.6** | 39.95 | Amazon |
| **Raspberry Pi Camera Module** | 21.1 | AliExpress |
| **Solar Panel** | 15.87 | Amazon |
| **18650 Battery Set** | 14.99 | Amazon |
| **MPU 6050 (IMU)** | 4.52 | Amazon |
| **Optical Encoders** | 11.95 | Amazon |
| **Car Chassis** | 15.99 | Amazon |
| **L298N Dual H Bridge [x2]** | 8.99 | Amazon |
| **Camera Case** | 6.99 | Amazon |
| **TP-4056 (Charge Controller)** | 4.61 | Amazon |
| **UBEC DC/DC Step-Down 5V Converter** | 9.95 | Adafruit |

Total cost with computer vision: $203.41

Total cost without computer vision: $126.82

We save $76.59 by not using the computer vision algorithm and solely using the GSA. For situations where high processing power and a camera are not necessary, and if equipped with better more power efficient motors, it would be economical to use GSA and use a model of the SunE-Bot that is not equipped with computer vision.

## Safety

A major safety hazard is the exposed circuit of soldering on the through-hole board which the microcontroller rests on. It is prone to shorting and the only insulation it has is the plastic it rests on. It is important that the robot and its circuitry are kept away from conductive materials and liquids. The prototype of the SunE-Bot is not designed to be at all water-proof.

Safety precautions must be taken with the 18650 Lithium Ion batteries. They are rated safe for a 500mA discharge rate which is within our robot’s use. However, precautions should be taken if more current draining peripherals are added. Batteries must be removed if they feel hot and should be kept away from temperatures above 70° C. The batteries must be inserted in the battery packs with the (+) and (-) poles in the correct orientation. This means that the (-) side of the battery must be inserted on the side of the battery packs with the spring. When not in use, as a precaution, the batteries should be removed from the robot and stored in a cool and dry place. Batteries should not be kept in direct sunlight.

As a safety precaution, the solar panel should be detached while the SunE-Bot is not in use. This will help in the longevity of the panel and the charge circuit.

## Performance, Security, Quality, Reliability, Aesthetics etc.

At its current prototype stage, SunE-Bot does not have many security features that we would implement in the final design. The Raspberry Pi is password protected, but it is possible for it to be hacked. In future versions we plan to incorporate higher security measures, especially when we plan to set the robot off on its own.

The aesthetics of our prototype do not give a clear representation of what we hope to show on the final design. We have exposed wiring and circuitry that not only pose hardware risks, but also look messy. We have taken measures to make our prototype look as good as possible through soldering, trimming and color-coding wires, and by matching the color of our 3D printed parts with the same yellow color that is used in the plastic of the wheels and motors. In the final design, we hope to encapsulate a printed circuit board with a sleek, closed body to give the SunE-Bot a modern futuristic look. Similar to our model, we would have a consistent color scheme.

When testing our robot, we wanted to ensure it functioned as reliably as possible given that it is meant to be an autonomous robot. Heavy testing time was put into modifying and improving our algorithms. Through our testing, we found that the success rate of our computer vision algorithm was roughly 95% while our traversal algorithms was closer to 80% due to its heavy reliance on IMU turning. Given more time we would have liked to improve the reliability of both these algorithms to closer to a 100% reliability.

## Documentation

Throughout this project our group has kept short records on - what parts have been bought, what parts have been tested, what problems we ran into, and how we fixed it - in log files on a weekly basis. We also kept recorded of our weekly priorities on a shared Google Doc. This documentation helped as a reference and checklist to what had been accomplished and what still needed to be done as each week went by. Additional design diagrams for state machines and circuit designs were drafted in a notebook before reformatting onto a final design on a CAD software.

The three of us also worked on the same files of code throughout this project and we were able to control version updates through Github. The newest most updated version was always on Github - and we pushed and pulled our code from the server accordingly whenever one of us made modifications. If more than one person was editing the code at the same time, we would have one stash the changes or push the codes into the “temp” branch while the other pushed to the “master” branch to avoid conflicts.

## Risks and Volatile Areas

One especially risky area of our power-charging circuit was the connection between the solar panel and the battery. This circuit had to be handled carefully because it was a free connection that flowed uncontrolled current from the solar panel to the battery. Risks include: a reverse current may flow from the battery to the solar panel if the panel is not receiving enough solar power and the battery is at a high voltage causing the reverse voltage threshold of the Schottky diode to be surpassed. Also, the battery can only receive a certain threshold of voltage called the “trickle charge voltage” or the battery can run a risk of over-charging and may be severely damaged. We ensured that the Schottky diode we used to have a high enough threshold voltage that we do not need to seriously worry about reverse current flow. We also included a step-down converter and charge controller between the panel and the battery to prevent overcharging. The charge controller turns on a green LED when the battery is fully charged.

# Experiment Design and Feasibility Study

## Experiment Design

1. **Basic Solar Panel/Photoresistor Experiments**

Conducted by Chris/James

**Objective:** Find the range of readings for solar panel and photoresistors indoor/outdoor environments.

**Setup:** Construct the voltage divider circuits for solar panel and photoresistors, connecting outputs to ADC reading of Teensy. Attach charge controller to voltage divider or solar panel to simulate similar load conditions to when SunE-bot is operating.

**Procedure:** Place bot in different typical operating light conditions, collect analog data reading values. For outdoor testing, place the robot on the ground in full, direct sunlight and adjust panel/photoresistor orientations to test various light strength readings. For indoor testing, use a flashlight at different distances from sensors to test the range of readings and then do the same for the robot placed on the floor without a flashlight.

**Expected Result:** Sensors should be able to sense a wide range of light strengths. This test should give us feedback on which photoresistor is more sensitive, if any. We should see higher readings for brighter areas and when the flashlight is applied.

1. **Angle-Biased Photoresistors**

Conducted by Yidi/Chris

**Objective:** Test if Photoresistors are sensitive enough to recognize the direction of the light source

**Setup:** Create photoresistor circuit, with the four photoresistors attached through a voltage divider. Teensy ADC ports. Photoresistors should be spaced evenly apart at each corner of the solar panel and not be obstructed by any shadow created by the robot or the surrounding environment. They must be placed at an angle to bias the data of the light reading.

**Procedure:** From different angles, shine the flashlight on the photoresistors. Observe the readings and see if they properly correlate with the direction the light is being shined from.

**Expected Result:** Photoresistors should be relatively equal in sensitivity when light is shined from directly ahead. The readings should be higher on the side that we are shining the light source from.

1. **Encoders**

Conducted by Chris/James/Yidi

**Objective:** Ensure that encoder ticks are properly correlating with the distance traveled by the wheels.

**Setup:** Connect the optical encoder to our wheel axle with power and ground, connect the output signal to and oscilloscope and a teensy digitalRead pin in series and common ground both. Setup software in Arduino to print and increment a tick count. Connect the Teensy to the computer through the USB to read the results.

**Procedure:** Turn the wheel one full rotation and compare the expected result with the ticks recorded by the software and a count of number of ticks displayed on the oscilloscope.

**Expected Result:** By counting 20 holes in the optical encoder and 21 cm per wheel rotation, we calculated each tick should measure 1.05 cm. One full wheel rotation should output 20 ticks on the software and oscilloscope.

1. **Solar Panel Charging**

Conducted by Chris/James

**Objective:** Test if enough current can be produced to charge battery.

**Setup:** Prepare two solar panels:  a 5V, 3W solar panel and a 12V, 5W solar panel.

**Procedure:** Under room conditions and outdoors, connect solar panel to voltage divider and charging circuit and measure current at output as well as the current going from the charge controller to the battery.

**Expected Result:** Under sufficient lighting, we expect to see current passing from the solar panel to the step-down converter when it is higher than its minimum input voltage as well as approximately 500mA of current flowing from the charge controller to the battery.

1. **Turning: Encoders**

Conducted by Chris/James

**Objective:** Determine if encoders can be used to accurately turn precise angles.

**Setup:** Attach encoders to back two wheels, with encoder sensors installed and attached to the Teensy.

**Procedure:** Count encoder ticks as robot turns various angles Check if ticks are proportional to angle turned.

**Expected Result:** Encoders should produce a consistent number of ticks with a consistent angle turned.

1. **Turning: Delay**

Conducted by Chris

**Objective:** Encoder turning was not viable. Determine if delay time (waiting a set time) and movement in very short bursts is viable for accurately turning precise angles.

Setup: Basic robot setup.

**Procedure:** Program robot to turn with a set delay in long and short bursts. See if the angle is consistent and accurate.

**Expected Result:** The robot should turn consistent amounts during consistent durations of time.

1. **Turning: Compass**

Conducted by Chris/Yidi

**Objective:** Encoder/Delay turning were not viable. Determine if a compass is accurate in turning precise angles.

**Setup:** Connect compass to Teensy, with basic robot setup.

**Procedure:** Have the compass calibrate and get an initial angle reading. Have the robot turn and stop when compass reads it has passed a certain angle. Check with various angles if compass-determined angle is consistent with actual angle turned. Perform the test indoors and outdoors to determine if the electromagnetics would be affected by the metals indoors.

**Expected Result:** The compass should accurately identify when the robot has turned a certain angle.

1. **Turning: IMU**

Conducted by Chris/Yidi

**Objective:** Encoder/Delay/Compass turning were not viable. Determine if an IMU is accurate in turning precise angles.

**Setup:** Connect IMU to Teensy, with basic robot setup.

**Procedure:** Have the IMU calibrate and get an initial angle reading. Have the robot turn and stop when IMU reads it has passed a certain angle. Check with various angles if IMU-determined angle is consistent with actual true angle.

**Expected Result:** The IMU should accurately identify when the robot has turned a certain angle.

1. **Raspberry Pi Camera Testing**

Conducted by Yidi

**Objective:** Test if the computer vision algorithm could effectively detect the ellipse-fitting light spots and ignore disturbances.

**Setup:** Connect Raspberry Pi and camera, powered and functioning. Establish data connections between the RPI and the Teensy. Program Teensy to operate the computer vision algorithm with input from the Raspberry Pi.

**Procedure:** Capture the raw video with an RPI camera. Use Ellipse-fitting to detect and isolate flashlight spot from other brightness spots.

Expected Result: The camera and RPI should be able to detect and isolate the brightness spots.

## Experiment Results, Data Analysis and Feasibility

Carry out the experiments designed above and present experimental data, quantitative analysis of the data, and the conclusion to show the feasibility of your project idea, how the experiments help you decide whether your technical design objectives can be achieved, and how they help you select the best solution to be further developed in the design project.

State clearly who is responsible for which task

**Basic solar panel/photoresistor experiments**

Conducted by James/Chris

For indoor testing, our photoresistors showed a range from about 196-252 under ambient light and higher values of 369-491 under the flashlight. Our solar panel readings showed readings of 92-104 for ambient lighting and 110-157 under the flashlight. For outdoor testing the photoresistors showed a range of 329-512 in the shadows and 831-920 in direct sunlight. Our solar panel showed a range of 136-144 in the shadow and 165-184 in direct sunlight.

**Angle-Biased Photoresistors**

Conducted by Chris

The data showed that the right photoresistor readings were significantly higher than the left, even under similar light conditions. However, there was a significant increase to both sides which correlated with the angle of the flashlight, which indicated to us that they were sensitive to the direction and angle of the light source. We were able to account for the slightly higher right readings by multiplying the right side by a constant of 1.3.

**Encoders**

Conducted by James/Chris

In the first stages of testing encoders for movement, encoder ticks would be very inconsistent with movement distance. Encoder tick count would vary from 20 to 60 for every 30 cm travelled and were hundreds of ticks higher than what was expected from the distance. Although proportional, the number was nowhere near to the 20 ticks per wheel rotation that we expected to see. Eventually, we added capacitors to filter out the noise. After this adjustment every 30 cm would consistently be ~39 ticks which is what we hope to see. We determined movement with encoders to be consistently viable and we used this method in our algorithms.

**Solar Panel/Charging Circuit**

Conducted by Chris/Yidi

The solar panel produced ~400mA when connected to the charging circuit, and ~450mA from the charger to the 18650 batteries. This showed promise for charging the battery with our solar panel, and we calculated and tested the battery to fully charge in ~3 hours given the current flow. The circuit created in this experiment is used in the prototype.

**Turning: Encoders**

Conducted by James/Chris

We tested having the robot turn and stop when reaching certain encoder ticks. We primarily attempted to perfect the 90° turn, which through testing we determined to be ~26 encoder ticks. However multiple tests of 26-tick turn produced inconsistent results. The robot would turn roughly 90° with an error of 10° on either side. This was due to slippage of the wheels on the surface, producing incorrect readings of the encoders. Through this experiment we determined using encoder to turn was not a viable solution and we moved to delay turning.

**Turning: Delay**

Conducted by Chris

We tested having the robot start turning, then have the microcontroller wait for set times before the turning stopped. Through testing we determined having the microcontroller delay for ~980 ms produced a turn angle of 90°. Once again however the results were inconsistent: the delay was inaccurate and produced an error of ~5-15° on either side. We attributed this error to battery level. Delay turning is heavily dependent on the voltage of the battery, which determines the speed of the wheels when turning. This would change the angle turned if the delay does not change as well. Through this experiment we determined delay turning was not a viable solution and we moved to a compass.

**Turning: Compass**

Conducted by Chris/Yidi

We tested by having the compass get an initial angle reading before rotating the robot a certain angle, and checking the next compass angle reading. Performing this test several times quickly showed the compass was wildly inaccurate. Rotating the robot by an actual angle of 90° produced strange readings from the compass ranging from 90° to 40°. We discovered the compass was more accurate outdoors than indoors, which means the compass was most likely affected by possible electromagnetic forces and metals in the building. It was too dependent and sensitive to its location and we determined the compass was not a viable solution.

**Turning: IMU**

Conducted by Chris/Yidi/James

We tested the IMU in similar ways to the compass: by having the IMU obtain an initial angle reading before rotating the car manually and checking the IMU reading again. Unlike the compass, the IMU was not affected by the metal materials in its environment. It regularly produced accurate yaw readings and currently is the sensor used in our prototype.

**Raspberry Pi Camera Testing**

Conducted by Yidi

Testing the computer vision image processing and whether it was able to detect the flashlight beam involved shining it at various distances, off various surfaces to determine if the error rate was low enough to be viable and if the processing speed is fast enough to support real time image processing. Results showed that a minor delay was measured, and that the camera was able to accurately detect the flashlight beam in a timely manner. This was made even more accurate by modifying the maximum-contour-finding algorithm and threshold for brightness. We determined the camera and RPI to be viable for the Computer Vision Algorithm and we use these parts in our current prototype.

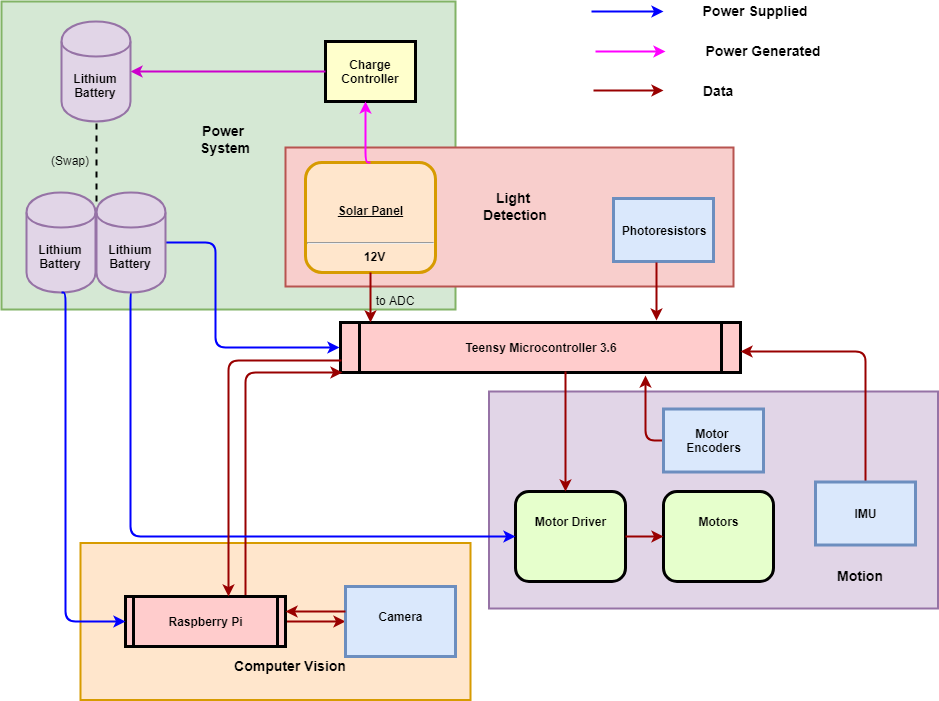
# Architecture and High Level Design

The architecture provides the top-level design view of a system and provides a basis for more detailed design work. These are the top-level components of the system you are building and their relationships.

## System Architecture and Design

We have 4 main systems operating in the SunE-Bot: the Light Detection system, the Power system, the Movement system, and the Computer Vision system. In general terms, Christopher handled the Light Detection and Power systems, James handled the Movement system, and Yidi handled the Computer Vision system. The Light Detection system consists of gathering data from the sensors and solar panel to serve as input for the Movement system. The Movement system controls the motors and operates the GSA and movement of the CVA. The Computer Vision system analyzes the video data and sends signals to the Motion system to articulate when it is oriented towards the brightest heading. Finally, the Power system delivers power to the computers and includes the solar charging of the battery. The systems can be seen in Fig. 1.

## Hardware Architecture



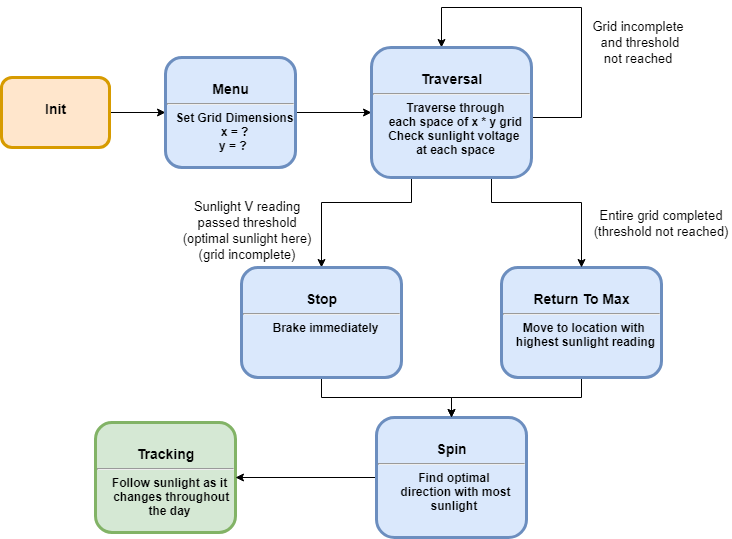
*Fig. 1: Hardware and System Block Diagram*

Above shows our system and hardware architecture design.

There are two battery groups in our system. The solar panel with the charge circuit is charging one battery. And another two batteries are providing power for the other components, including the Teensy 3.6, RPI and sensors. The power parts were mainly done by Christopher.

The Teensy 3.6 serves as the control unit for the robot’s decisions and motions. It sends signal to the motor drivers and reads data from the IMU, encoders, photoresistors, and solar panel. Just two signal lines connect the Teensy and the RPI, one for signal sent and one for signal received. The camera is only connected to the RPI and has no interaction with the Teensy.

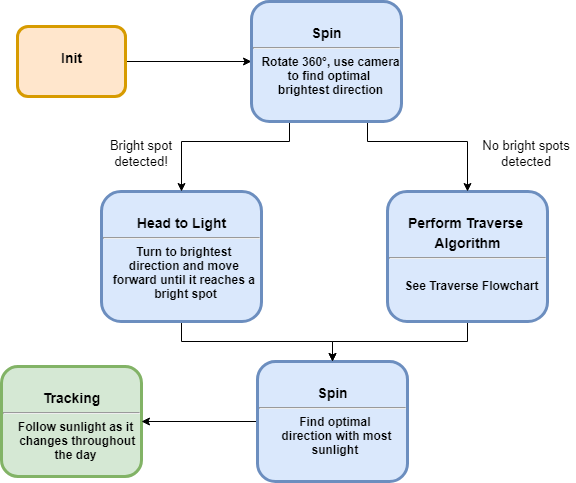
## Software Architecture



*Fig.2: Grid-Search Mode Algorithm*

This figure shows how the GSA operates.

At the startup, the robot is in the menu function until the search method and grid size (if GSA is selected) gets set. Once in GSA mode, the robot initializes its current coordinate to be the start-point (x=0, y=0), then starts traversing the selected area while continuously checking whether the sunlight is surpassing the threshold level, as well as recording the photoresistor reading for the corresponding (x, y) coordinate. If the light threshold is met, SunE-Bot should be in a satisfactory location and stops immediately and activate the Spin function. This causes it to rotate 360° and then return to the heading that yielded the highest voltage to the solar panel. At this point, the Tracking function activates, and the robot uses its photoresistors to monitor for any major deviation between its right and left photoresistor readings. If it reads that one side is significantly higher than the other, due to the angled bias of the photoresistors we know that the orientation of the light source is not directly ahead. SunE-Bot makes minor heading adjustments to the left or right to rematch the values of the photoresistors which serves as a function to track the movement of the sun. These are done by Yidi, tested and debugged by Christopher. The robot also monitors for large changes in light readings, which can signify a shadow being cast over its position. If this happens, the robot restarts the Traversal function, setting its current location as the new start-point. However, if during the Traversal function the sensors did not reach the threshold level, SunE-Bot instead returns to the position that yielded the highest sensor reading. The Traversal and Return to Max Functions were done by James. From there it activates the Spin and Tracking functions.



*Fig.3: Computer Vision Mode Algorithm*

This figure shows how the CVA operates.

When switched to computer vision mode, the Teensy sends a control signal to the RPI to start the camera preview. The robot does a 360° spin at a 10° resolution, during which the RPI camera detects the brightest light spots. Once a light spot is recognized, the RPI sends signal to the Teensy, and the Teensy records its current angle. If a stronger spot is detected, the RPI refreshes its max record and sends the signal again to the Teensy, and the Teensy refreshes the angle value. After the 360° spin, the SunE-Bot returns to the angle it recorded previously, then heads in the direction until light sensor reading threshold has been met. Finally, same as in the Grid-Search Algorithm, the Spin and Tracking functions are executed. The Raspberry Pi image analysis was developed by Yidi, and Christopher made the receiving end on the Teensy.

As for the computer vision implementation, GPIO (RPI.GPIO) and Camera module (picamera) libraries are used in our Python and OpenCV codes. If no control signal from the Teensy is received, the camera remains off for the sake of saving power. Initially, a brightness threshold is set to adjust the binarization of the image. Gaussian blur and Canny edge detection are used later to better extract the contour of the light spots. In each frame, only the max contour is recorded. If a stronger source spot is detected in the following frames, the recorded max value is updated and sends a signal to the Teensy. The computer vision part is all done by Yidi.

## Rationale and Alternatives

**Rationale**

The sensors (photoresistors and solar panel output voltage) we use for the GSA require to be directly at the position that they are being used to test. The upside of using sensors is that they give the most direct and reliable information on the light levels, which will not be confused by a false positive (e.g. the computer vision might recognize something such as a pair of white shoes for being a bright spot). We chose to use not solely the solar panel for reading light but also the photoresistors because we wanted to keep the solar panel angled to allow for more effective power generation, without requiring it to perform a 360° check at each grid coordinate. We chose to put a photoresistor on each corner surrounding the solar panel, so that the robot would know if any corner of the solar panel is shadowed. This is important because solar panels have a string connection throughout, so even blocking just one portion can severely hamper its generation. The GSA is useful because it provides the most accurate information of where in a certain area is the brightest.

However, GSA will be less efficient and economic if a relatively large area is required by the user, especially when the light spots are scarcely distributed. Under this circumstance, a more direct searching algorithm is necessary, and using computer vision would be the most straightforward solution. Computer vision makes our design free from the inherent limitations and make optimal spots outside of the pre-defined area accessible. However, although power being spent on movement is saved by heading directly to the light, powering the RPI and camera does draw more power. Still, most autonomous drones are likely equipped with a camera for data gathering, so it would be a waste not to use it.

**Alternatives**

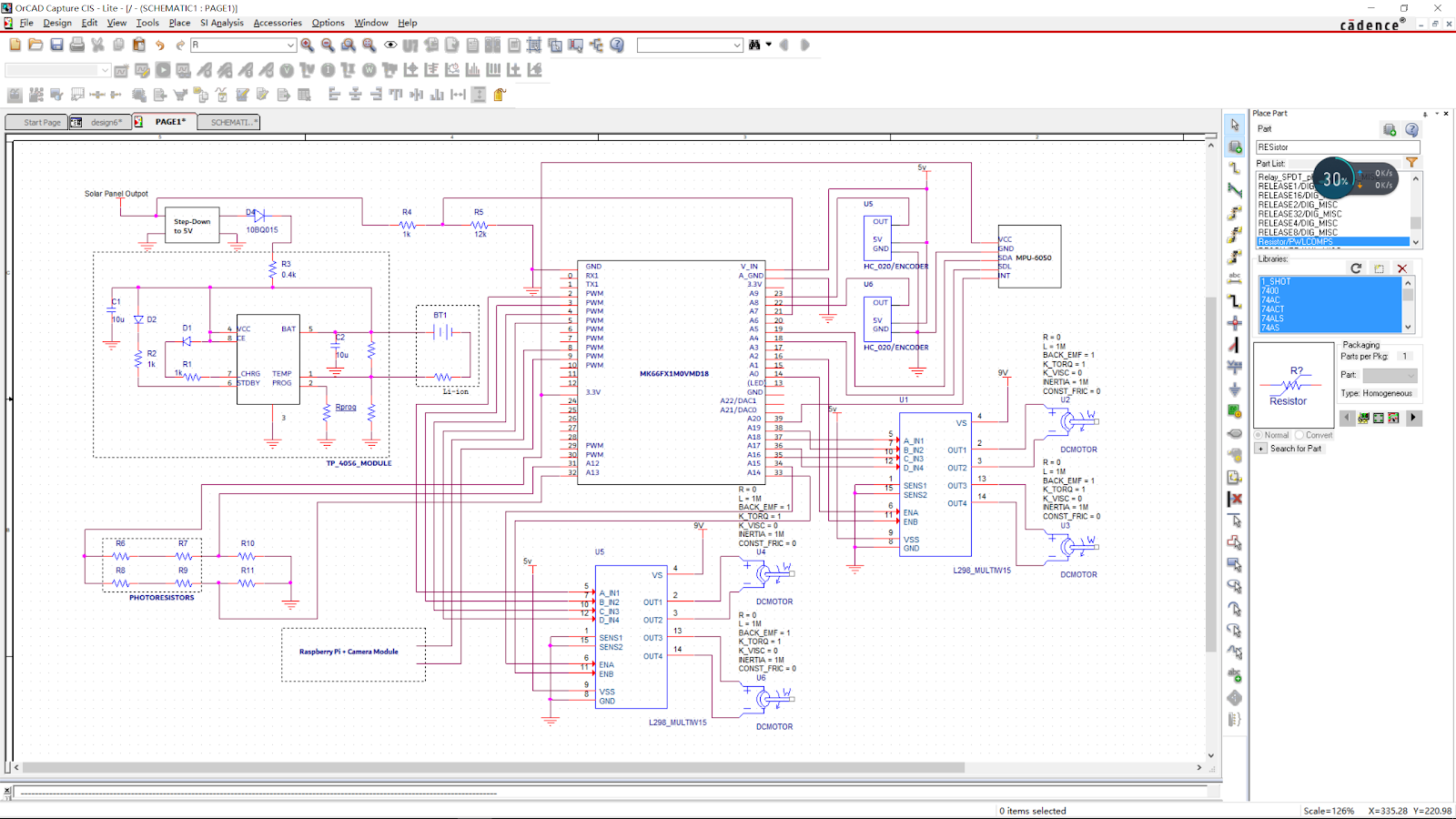
It is possible for us to rely solely on the solar panel for light readings and not use any photoresistors. This will require a solar panel that is motorized so that it can be parallel to the ground during traversal and then tilt after finding the desired coordinate, or a more complex and somewhat more ineffective GSA method. Since the solar panel is oriented at an angle, the readings would be biased, and the GSA would require additional steps to account for this bias. However, the performance of the CVA would not suffer much. The benefits of this include more body space on the robot without the sensors and sensor holders, a slightly lighter weight, less wiring, more pins available for use on the Teensy, slightly cheaper production costs, and a more direct reading of the voltage being generated by the light at each location.

An alternative light detecting sensors we could have used include infrared sensors and photodiodes. We wanted to stay away from infrared sensors because they do not work with fluorescent indoor lighting and the correlation between infrared levels and sunlight levels is not necessarily direct. We chose photoresistors over photodiodes because the circuitry they required was easier for us to implement.

We are processing video in the computer vision algorithm, and this is consuming higher power than if we just process images. The reason why we choose to process video instead of images is because we wanted to use continuous video to increase recognition accuracy -- we do not want to miss a spot. If we were going to take photos between small turns, the time window is small -- it is very possible that the RPI misses the image of the light spot. Although it would be harder to precisely measure and calculate the code-execution time, this approach might be worth trying for power-saving consideration.

# Low Level Design

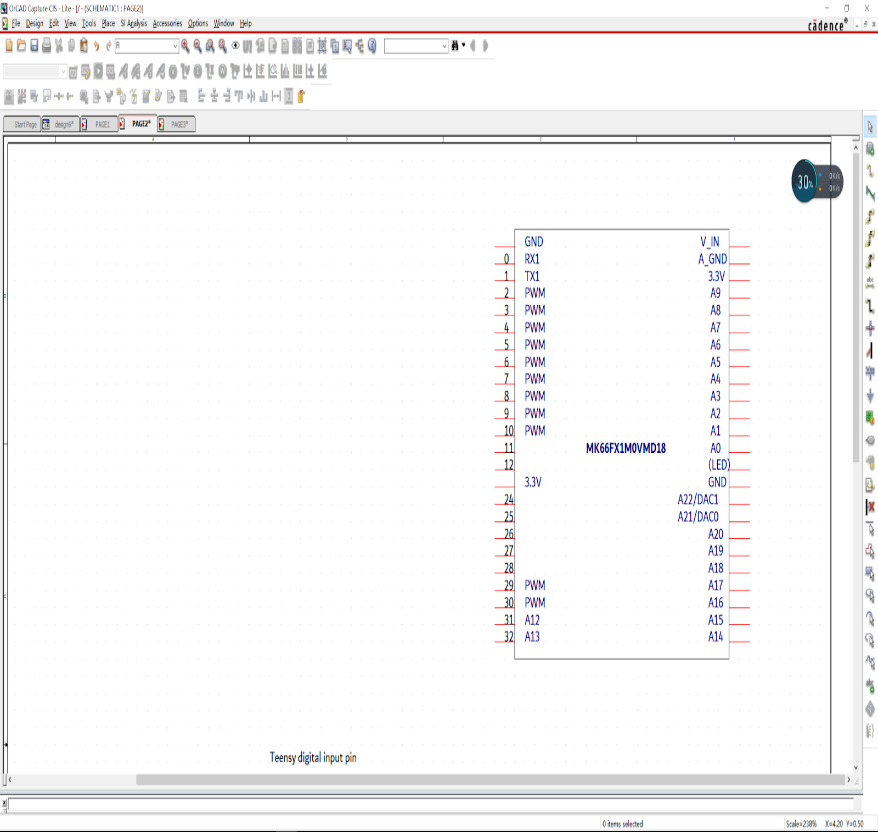
Below is a low-level diagram of all electronic components and parts of the SunE-Bot. All wiring and circuitry is shown, including all wiring between components and to VCC/ground.



*Fig.4: CAD of Low Level Design*

## Module 1: Teensy 3.6 Module

### Processing narrative for Teensy 3.6



*Fig.5: CAD of Teensy*

The Teensy USB Development Board is a complete USB-based microcontroller development system. Teensy 3.6 features a 32-bit 180 MHz ARM Cortex-M4 processor with floating point units. All digital and analog pins are 3.3 volts. We chose the Teensy 3.6 as our major control unit. All the decisions are made according to the calculation based on the data gathered by the peripheral sensors.

### Teensy 3.6 Interface Description

We are using a 3.7V voltage to power the Teensy. The wire comes out from the positive end of the battery is directly connected to the “V\_IN” pin of the Teensy, and all the components are common grounded. Besides the pins related to power supply, six types of pins on the Teensy are used to connect all the peripheral components: SDA and SCL, digital pins, analog-read pins, PWM pins, default LED pins and 3.3V output pins.

### Teensy 3.6 Processing Details

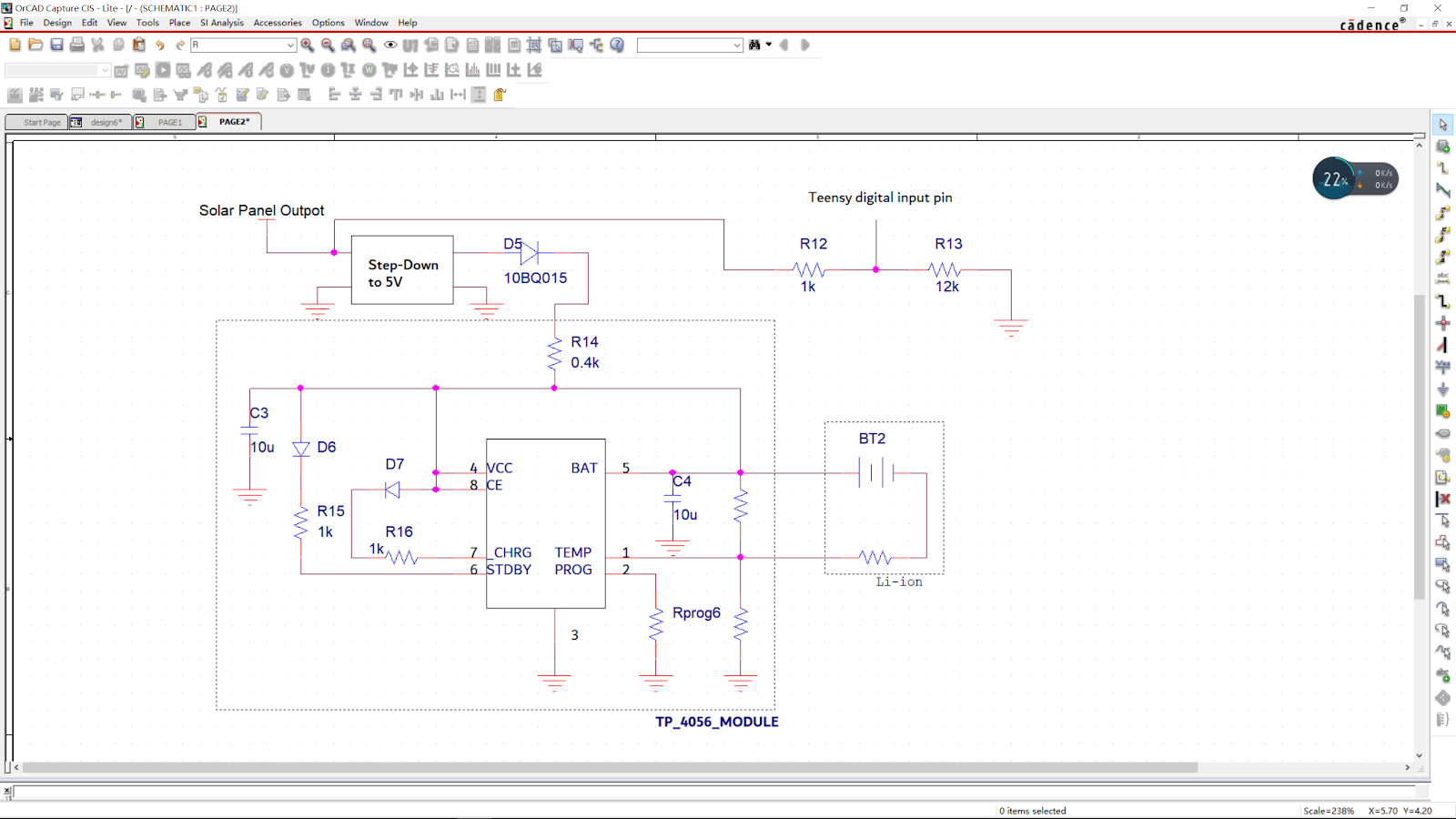
The Teensy collects data from sensors, including the IMU module, the encoders, the RPI and the camera, the photoresistors and the solar panel. In GSA, the Teensy will control the robot to do a traversal in the pre-defined area. In CVA, the Teensy will control the robot according to the signal from the RPI.



## Module 2: Charging Circuit Module

This part has been mainly done by Christopher.

### Processing narrative for Charging Circuit



*Fig.6: CAD of Charging Circuit*

This module is for battery charging. It converts light power into electric power, which is used to charge the battery.

### Charging Circuit Interface Description

About 8V to 14V voltages can be obtained by the solar panel. Using a voltage divider, the value directly goes into the Teensy to serve as a light intensity indicator. The same wire is connected to a step-down converter followed by a Schottky diode and a voltage regulator (TP-4065). The step-down converter provides a proper voltage level for battery charging, and the regulator can filter out unwanted ripples and smooths unwanted voltage spikes. The Schottky diode aims to prevent the current flow in a reverse direction under the circumstances in which the light is dim and the output from the solar panel and the charge circuit is lower than the charging battery’s internal voltage.

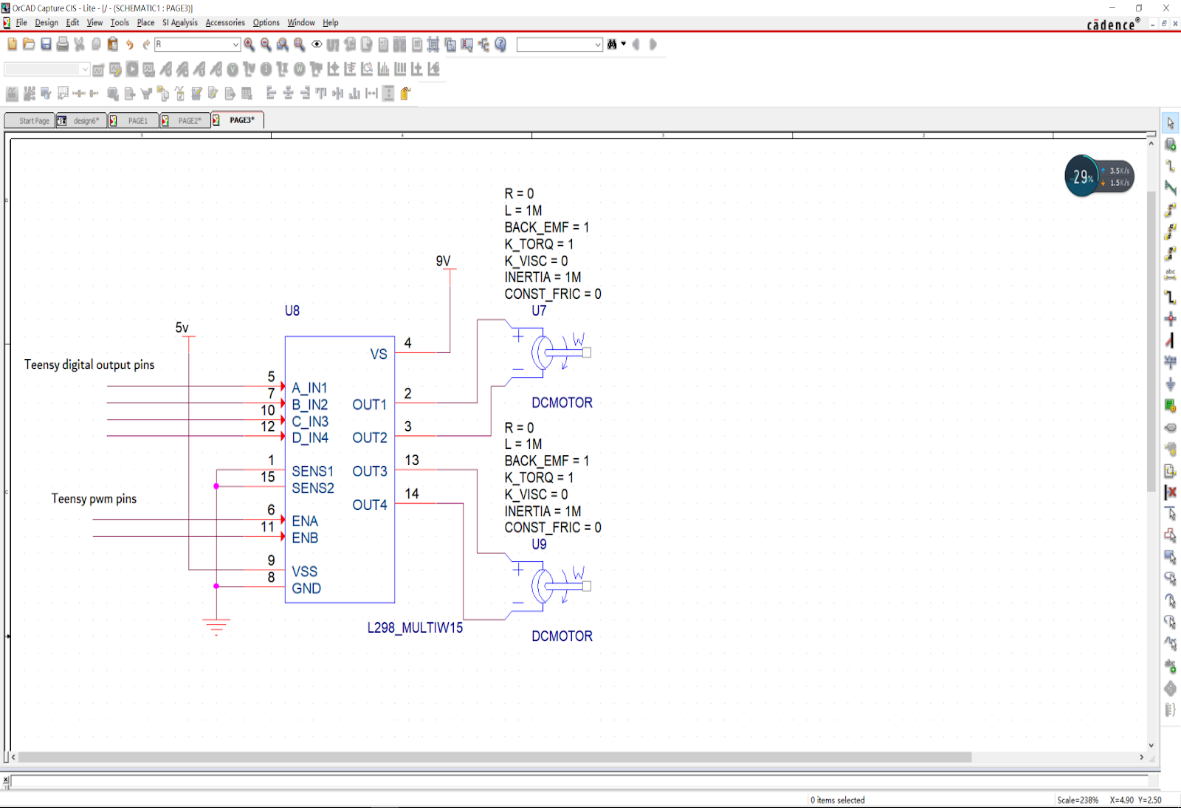
### Charging Circuit Processing Details

The Teensy and the charging circuit are in parallel to the output of the solar panel. The resistance of the digital input pin of the Teensy is high, so the current will mainly flow into the charging circuit which then goes into the battery. The voltage is divided by two resistors to reduce the voltage into the Teensy. In the Teensy code, digitalRead is called to get the raw voltage values. The voltage level from the solar panel to the Teensy proportionally reflects the light intensity using a lower voltage level.

## Module 3: Movement Module

This part has been mainly done by Yidi.

### Processing narrative for Movement



*Fig.7: Cad of Movement Module*

All the robot behaviors are based on movements of wheels according to the Teensy’s decision making results.

### Movement Interface Description

Two motor drivers (L298N) are used to control and power our four-wheel robot. Each motor driver controls the speed and rotation of two motors. Two wires from the Teensy digital outputs are connected to IN pins for direction control, and one wire from the Teensy PWM outputs is connected to the EN pin for speed control on the motor driver. Additionally, the motor drivers need two voltage inputs: a 3.3V~5V input as digital reference and a 7V~12V input for power supply.

### Movement Processing Details

The function analogWrite is called to set the current flow into the motors through the motor driver, and digitalWrite is called to set the rotating directions of the motors.

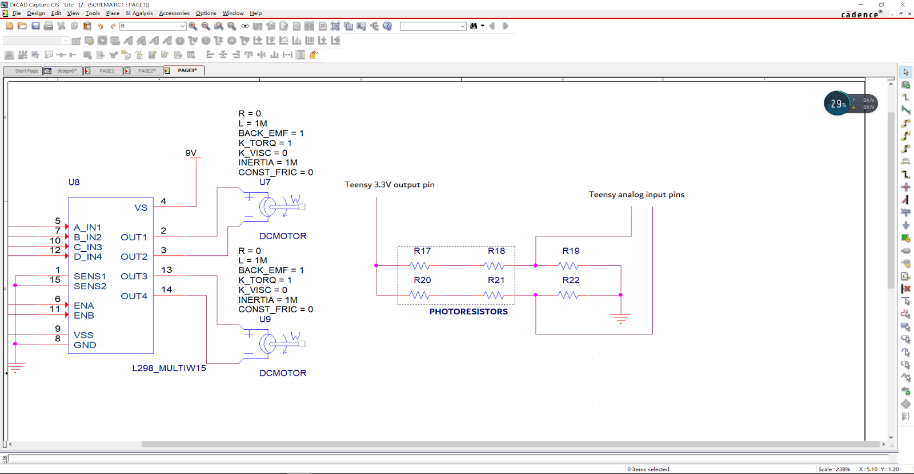
The quality of the motors we are using are not ideal, and inherent differences between them influence the performance of the whole robot. We found that one motor is significantly stronger than the others. Substantial time was put into adjusting the PWM to allow the robot to move straight.

The wheels are made of a hard and smooth plastic, which causes them to often slip on smooth ground when turning. This made encoder tick calculations while turning very inaccurate.

## Module 4: Light Sensor Module

This part was mainly done by Christopher and Yidi.

### Processing narrative for Light Sensor



*Fig.8: CAD of Light Sensor*

Besides the readings from the charging circuit, the Teensy obtains surrounding information from four other sensors: photoresistors, encoders, IMU, and RPI camera. When the light strength changes, the resistance of the photoresistors changes significantly, which makes them ideal for determining light intensity.

### Light Sensor Interface Description

Two photoresistors and a regular resistor are installed serially on either side of the SunE-Bot. With this setup we can get the average through hardware designing instead of increasing computing time in the MCU. The node between photoresistors and resistor is connected to an analog reading pin of the Teensy and the light intensity changes can be supervised accordingly.

### Light Sensor Processing Details

The photoresistors are grouped by the left side and the right side of the robot. The photoresistors are connected in series with the two voltage dividers, so our ADC readings gives us a proportion of the average voltage on each side. The brighter the light is, the lower the resistance, hence the higher the reading produced by the voltage reading. We use this raw value to understand the amount of light shining on the SunE-Bot.

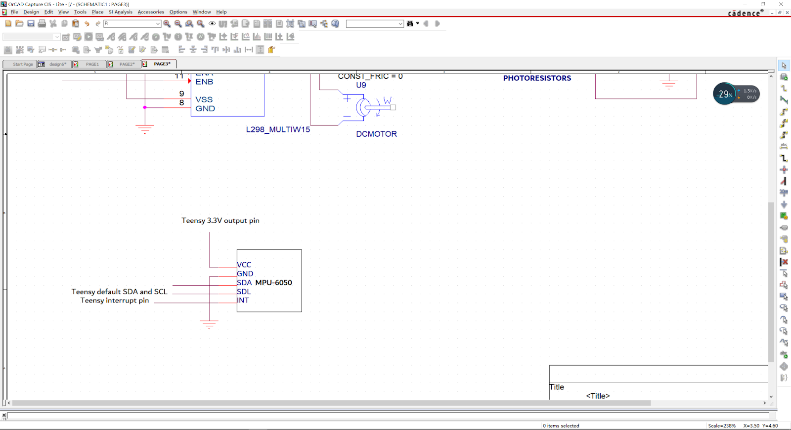
## Module 5: IMU Module

Mainly done by Christopher.

Calibration codes found and overflow issue solved by Yidi.

Debugged by James.

### Processing narrative for IMU



*Fig.9: CAD of IMU*

We are using the IMU module for precise turning. There is an accelerometer inside of the module. The communication between this module and the Teensy is based on I2C protocol. The relative angles of the robot heading can be obtained by a set of conversion and calculations from the raw data, and all the calculation functions and register definitions are written in a C++ header file in the Arduino library.

### IMU Interface Description

The pins we are using of the IMU are VCC, GND, SDA, SCL and INT. SDA and SCL are for the I2C communication and are directly connected to the default SDA and SCL pins of the Teensy. The data is transmitted when the interrupt is triggered (INT), and the INT pin is connected to a digital pin of the Teensy. The pin is declared using the attachInterrupt function. Any digital pin of the Teensy supports interrupt.

### IMU Processing Details

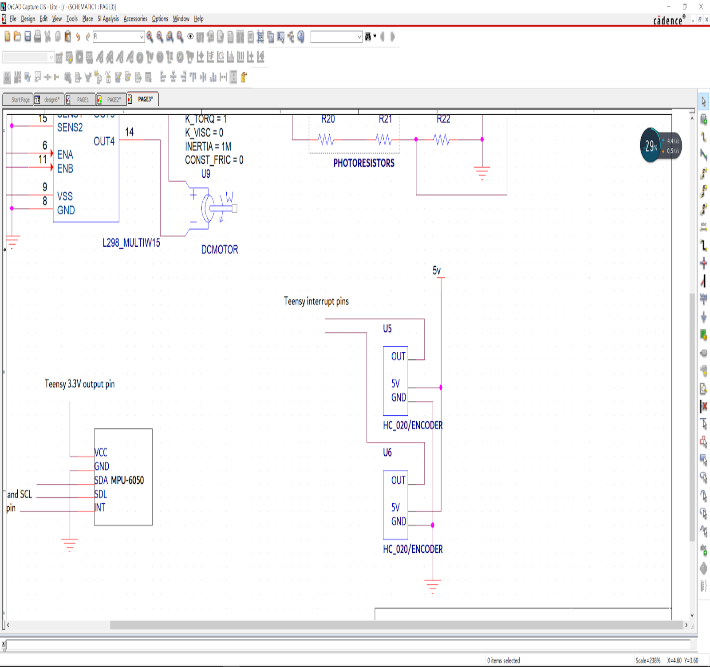
In our algorithm, after setup, the robot will wait until the IMU readings become stable. The yaw range is from -180 to 180°. In the precise turning functions, the initial yaw (°) value from IMU is recorded. The robot then begins turning and continues until the destination yaw angle is met.

## Module 6: Encoders Module

Tested by James, Yidi and Christopher.

Implemented by James

### Processing narrative for Encoders



*Fig.10: CAD of Encoder*

A motor encoder is a rotary encoder mounted to an electric motor that provides closed loop feedback signals by tracking the speed and/or position of a motor shaft. There are 3 pins on each of the encoders: VCC, GND and OUT. The OUT pin is for interrupt.

### Encoders Interface Description

The interrupt pins of both the encoders are connected to two digital pins of the Teensy. The pins are defined in the code using the attachInterrupt function.

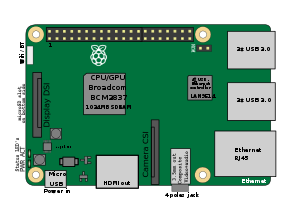
### Encoders Processing Details

Every time a new pulse generated by the encoder, an interrupt will be triggered in the Teensy and the Interrupt functions will be executed, in which we increment the encoder ticks for the left and the right wheels. We are using the encoders for distance measurement in the GSA for self-navigation as well as for navigating the menu.

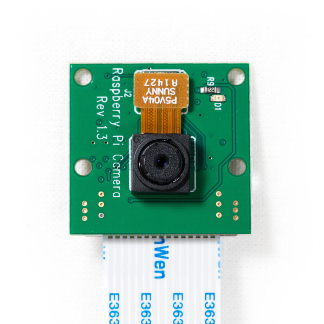
## Module 6: Raspberry Pi and Picamera Module

Mainly done by Yidi

### Raspberry Pi and Picamera Processing Details



*Fig.11: Graphic of Raspberry Pi 3*



*Fig.12: Picamera*

These parts are for computer vision, including video collecting and image processing. The surrounding environment information is gathered by the camera while the robot keeps spinning. The RPI processes the information in real-time and determines the direction of the light spot. The signal sent from RPI to the Teensy gives the Teensy the information needed to make the decision and spin to the correct direction.

### Raspberry Pi and Picamera Interface Description

There are two GPIO pins connecting from the Teensy to the RPI. One receives control signals from the Teensy to indicate when to turn on the camera. The other pin sends signals to the Teensy when a qualified light spot is detected. Both GPIO pins of the RPI are connected to the digital pins of the Teensy separately.

### Raspberry Pi and Picamera Processing Details

The camera will first keep sleeping until it receives the START signal from the Teensy. The RGB video gathered by the Pi camera will be first converted to gray-scale. A set of boundary values are set to distinguish brighter parts in the video. Then the Canny edge detector is used to extract the contours of the brightness, and only the maximum contour in one frame will be kept for comparison to the recorded maximum.

# Technical Problem Solving

Document all the technical difficulties you encountered

## The Power Buses Problem

Throughout the early to mid-stages of our project, we frequently ran into mysterious glitches that caused the SunE-Bot to randomly stop moving or restart. It took a while for us to pinpoint exactly what was causing this, so it caused severe delays to the progress of our project. By using multimeters, we found that the breadboard we had been using for our microcontroller and power buses had numerous poor connections.

## Solving the Power Buses Problem

Before we were confident enough in our design that we felt ready to solder the components to a through-hole board, our only solution to combating the poor breadboard connects was to reposition the pins into different holes. However, the breadboard was fairly small, and we had very limited space and with so many poor connections we had a hard time of finding spots that did not have any issues. We switched to a new breadboard which served us better, but from there we found that there were still problems with the power switch circuit that we had soldered Eventually, we decided it was necessary to upgrade from breadboards completely so we soldered all of our components and re-soldered the power switch circuit and the SunE-Bot stopped having connection issues.

## The Charging Problem

In the planning stages, we did research and found circuits that were designed to both charge and drain the same battery using a combination of diodes. We tried using a few versions of this circuit, but we could not effectively get the battery to charge while simultaneously using it to power our Teensy.

## Solving the Charging Problem

We chose to keep the charging circuit separate from the power system used to power the motors and the microcontroller. We used a diode to prevent back feeding with a step-down converter to lower the solar panels output voltage to be low enough to pass into the TP4056 charge controller.

## The Encoder Problem

Our encoders were seemingly working as they showed positive feedback with rotation of the encoder wheel. However, the distance traveled by the wheel was much less than we expected to see based on the distance per tick and there was a large variation between tick counts for similar distances. It showed over 400 ticks per wheel rotation when we expected to see approximately 30. By connecting our encoder signal to an oscilloscope, we saw that it was properly showing 30 high and low signals, but our microcontroller was still reading hundreds.

## Solving Encoder Problem

We determined that there was some sort of bouncing problem with the encoders. The oscilloscope seemed to display high oscillation at the start of each wheel rotation. To combat this, we added a slight 30ms delay before we started registering the ticks and accounted for it by adjusting the total tick count. We still had unproportional tick-to-distance ratio which we tried to solve through other software solutions, but we had no luck. Through online research, we found that capacitors were an effective solution to encoder noise creating false-positive readings, and by adding them we no longer had false-positives and the tick-to-distance ratio adjusted to exactly what we expected to see based on our measurements.

## The Turning Problem

Despite resolving the false-positives with our encoder readings, the turns our robot made were not precise enough to implement the GSA. This was because the encoders could not account for any slippage between the wheels and the ground. We tested delay-based turning, using time increments as the measurement for how much the wheels turned but still faced the same problem. We added the LSM303 digital compass to have direct feedback of how much the car turned. However, despite calibration, this compass was far too susceptible to electromagnetic interference.

## Solving the Turning Problem

The turning problem was solved by adding the MPU 6050, which is an inertial measurement unit. It provided us with an accurate enough yaw reading which we were able to use as feedback for how much our robot turned.

# User Interface Design

## Application Control

Our robot is designed to function mostly autonomously, so we try to keep the user interfacing to a minimum. Currently, we have no screen, so we use our buzzer as feedback for our menu which we navigate through using our wheel encoders.

## User Interface Screens

The robot’s user interface was designed by Chris and is controlled through wheel rotation with our encoders and our light sensors as buttons. The menu is navigated through using the audio outputs from our buzzer. In the current iteration of our software, there is two menus. The initial menu is to select the search algorithm type with the two options being the GSA and the CVA. It starts on the CVA option which is denoted by the high-pitched “chirp” noise. By rotating the wheel over 25 encoder ticks, the lower “boot tone” noise, will sound to signify the selection of the GSA. The menus can be continuously cycled through by rotation of the back right wheel. To confirm the selection, the user can use their fingers to cover the two front light sensors. The high-pitched “laser tone” should sound immediately. When CVA is selected, the robot will play a musical scale to indicate its setup configuration. It should be set down during this time as it will start running the algorithm soon. If GSA was selected, the grid size menu will start. Similarly, to the algorithm menu, it is navigated using the back right wheel and confirmed by covering the two front photoresistors. There are currently four grid size options to choose from, 2x2, 3x3, 4x4, and 5x5. The size selection is indicated by the number of beeps in quick succession as well as by their pitch. The two beeps in quick selection indicates the 2x2 grid, three beeps for the 3x3 grid, and so on. After confirming the selection, the robot should be set down as it will start moving after the automatic initialization is completed.

# Test Plan

## Test 1: Computer Vision Bright Spot Detection

Conducted by Yidi

**Objective**: After this test, we can determine the accuracy of the computer vision algorithm and properly set the parameters and thresholds.

**Setup:** The RPI is connected to a monitor through HDMI for easy visualization of the camera previewing video. The RPI’s output GPIO pin is connected to a LED, which is an indicator of signal sending. Many print functions are added into the python code for easy supervision of the program behaviors.

**Procedure:** Random light spots created by the flashlight are distributed on the ground. Manually adjust the position of the camera to detect light spots. Monitor the camera preview and LED to see whether a signal is sent when a new max spot is detected.

**Expected results:** Every time a new light spot is displayed on the monitor, the LEDs are supposed to blink and the new max contour value is printed to indicate that a new max spot has been detected and a signal is sent.

## Test 2: Communication between the RPI and the Teensy

Conducted by Yidi / Chris

**Objective**: after the test, we are able to see whether the signals transmitted between the RPI and the Teensy can be processed on time.

**Setup:** Set the video frame number to be 20.Set the RPI only start camera preview when it gets the control signal from the Teensy.Set the Teensy record the current yaw value when it gets the signal from the RPI.

**Procedure:** Start both the RPI and the Teensy. The RPI is waiting for the Teensy’s control signal, or it will not open camera. The states are printed onto the terminal. After the RPI get the signal, turn on the flashlight, shining on the ground in front of the RPI camera. Once the spot is detected, to check Teensy’s serial monitor to see whether the signal from the RPI has been received.

**Expected Results:** The RPI continue checking control signal from the Teensy every 10 seconds.The RPI opens the camera after Teensy’s setup finished.The Teensy gets a signal from the RPI when a new max light spot is found.The current yaw value is recorded according to the IMU reading by the Teensy.

## Test 3: Spinning and Tracking

Conducted by Yidi / Chris

**Objective**: after this test, we should be able to know whether the robot will return to the angle that it recorded to have the max light intensity, and whether the robot will keep tracking the light when the light is changing to a tilt angle.

**Setup:** Set the Teensy to only execute spin and tracking in the main loop. Use the flashlight to simulate changing sunlight.

**Procedure:** When the robot is doing spinning, turn on the flashlight, shining onto the solar panel, to see whether a certain sound is produced as an indicator that the solar panel reading reached a new max. After the spinning has been done, change the flashlight angle, shining on the photoresistors, to see whether the robot is able to track the light according to readings from photoresistors.

**Expected results:** Every time when we see the flashlight shine on the solar panel, if it is a new max, a certain sound will be produced, and the current angle will be recorded.After the spinning has been done, the robot stops where it starts spinning.When the angle of the flashlight changed, the robot are able to track.

## Test 4: GSA

Conducted by James

**Objective**: The objective of this test is to determine whether our robot would properly implement the Grid Search Algorithm

**Setup:** Set the robot on a tiled floor, and have it run the algorithm with a set grid size.

**Procedure:** Measure if actual robot grid path matches grid size programmed into the code. The closer the actual robot grid path matches the size the more accurate the program is.

**Expected results:** The robot should traverse through the grid path with minimal error.

## Test 5: Return to Max

Conducted by James

**Objective**: The objective of the test is to determine if the Return to Max function works correctly.

**Setup:** Set the robot on a tiled floor and run the GSA with a set grid size.

**Procedure:** Apply the flashlight to different spots on the grid and measure how close the robot finishes to the brightest spot. The closer the robot is to the bright spot the higher the accuracy of the program.

**Results:** The robot should find the brightest spot >95% of the time and should finish no more than 15 cm from the spot.

## Bug Tracking

Here is a record of our most severe bugs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Overview** | **Bugs** | **Assigned To** |
| **1** | Computer vision bright spot detection | 1. False positive triggered by white shoes  2. Bright spot not registered. | Yidi |
| **2** | Communication between RPI and Teensy | 1. Start signal received by RPI before sent by Teensy | Chris |
| **3** | Spinning and Tracking Test | 1. In both functions, the angle was not consistently the correct one.  2. In the tracking, the bot is less sensitive to the light source when it was at right | Chris/Yidi |
| **4** | GSA Test | 1. Angular turns were inaccurate throughout GSA test. | James |
| **5** | Return to Max Test | 1. Angle of return was correct, but distance traveled came up short. | James |

## Quality Control

In quality control, we try to resolve the bug and denote whether the resolution was successful.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case** | **Overview** | **Bugs** | **Modifications** | **Result** | **Date** |
| **1** | Computer vision bright spot detection | False positive triggered by white shoes | Slightly increased the bottom range of the qualified detected contour | Failed | Feb 26, 2018 |
| Bright spot not registered. | Doubled the delay of teensy for longer processing time for the RPI | Passed | Feb 26, 2018 |
| **2** | Communication between RPI and Teensy | Start signal received by RPI before sent by Teensy | Initialized the signal from the Teensy to “0” in the Teensy’s setup.  Made the RPI continue waiting while the signal did not be received. | Passed | Feb 23, 2018 |
| **3** | Spinning and Tracking Test | In both functions, the angle was not consistently the correct one. | Slightly increase the resolutions of spinning, with ticks of each turn unchanged. | Passed | March 7, 2018 |
| Change the threshold of photoresistors’ reading in the tracking. | Failed | Mar 8, 2018 |
| In the tracking, the bot is less sensitive to the light source when it was at right | Slightly increase the sensitivity of the right pair of photoresistors.  Slightly reduce the ticks of each turning in the tracking. | Passed | Mar 12, 2018 |
| **4** | GSA Test | Angular turns were inaccurate throughout GSA test. | Added IMU offsets calibration.  Slightly reduce the turning angle linearly. | Passed | Mar 8, 2018 |
| **5** | Return to Max Test | Angle of return was correct, but distance traveled came up short. | Modified calculations for distance determination which fixed the issue. | Passed | Feb 16, 2018 |

## Identification of critical components

**Computers**

* Teensy 3.6: The Teensy is a USB-based microcontroller which uses a 32-bit 180 MHz ARM Cortex-M4 processor with floating point unit. It has both digital and analog pins which must be properly registering and outputting signal.
* Raspberry Pi 3 Model B: The Raspberry Pi 3 is a single-board compact computer with a Broadcom BCM2387 chipset and a 1.2GHz Quad-Core ARM Cortex-A53. It has a 64-bit CPU 1GB of RAM and 40 GPIO pins. The RPI must be properly interfacing with the camera and that the GPIO pins can properly output and input signal.

**Peripherals**

* CYT1037 - A dual shaft gear motor produced by Clyewet. We used four of these, one for each wheel.
* L298N - A motor driver. that can control the current amount and directions of the motor. We used two of these, one for a pair of wheels.
* HC-020K - A photoelectric encoder with a single signal output. We used two of these, one on each back wheel.
* MPU 6050 - A six-axis IMU sensor which provides us three readings coming from the accelerometer and three readings from the gyroscope.
* Solar Panel - A solar panel absorb sunlight as a source of energy to generate electricity to charge a battery. and we also get light intensity information from it via ADC voltage readings.
* Photoresistors - Photoresistors are light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity. We use them as light intensity indicators.

## Items Not Tested by the Experiments

**Outside Test**

Up to now, all our tests of the algorithms have been done under artificial light source indoor. We have tested only sensors readings and solar panel charging outside. When we tried operating our robot outside, we found that the friction caused by the cement and concrete ground against our hard-plastic wheels caused too much of a disturbance to the movement of our robot. When we attempted turns, the entire robot would severely shake which could have been harmful to its frame, chassis and wiring, which was already prone to poor connections.

We tested robot movement once outside. The robot shook violently on the rough surface of the cement and the readings self-positioning sensors became extremely unreliable due to the vibration.

**Computer Vision Radius**

Since we only tested algorithms indoor, and the room space is limited. We do not know what should be the maximum distance at which the light spot is located and that could still be well detected by the Picamera.

**Actual Charging Time Under Real Light Conditions**

Because the charging time is massively affected by the strength of light to the solar panel, it is difficult to accurately determine charging time when the SunE-Bot is in a real setting with dynamic light conditions. In our experiment we used an ideal model to find how fast the SunE-Bot could charge its battery in optimal conditions.

# Test Report

## Test 1: Computer Vision Bright Spot Detection

Done by Yidi

**Test results:** Every time a new light spot displays on the monitor, the LED blinks and the new max contour value is printed to indicate that a new max spot has been detected and a signal sent.

**Comparison:** The test results are as expected.

**Analysis:** From the test results, we could conclude that our video processing algorithm is effective and efficient enough for light spots detection.

**Corrective actions taken:** None.

## Test 2: Communication between the RPI and the Teensy

Done by Yidi and Christopher

**Test results:** The RPI continue checking control signal from the Teensy every 10 seconds, until the Teensy finished setup. The Teensy gets a signal from the RPI when a new max light spot is found. The current yaw value is recorded according to the IMU reading by the Teensy, although lacking precision. The camera became less sensitive when we reduce the delay time between two slight turnings during spinning.

**Comparison:** The communication parts perform well. We casted 5 light spots on the ground. All of them had been successfully found by the camera when we made a 1 second delay between two slight turnings during spinning. And all signals had been received by the Teensy. In another test, we casted 5 light spots on the ground. Only one of them had been successfully found by the camera when we made a 0.3 second delay between two slight turnings during spinning. And all signals had been received by the Teensy.

**Analysis:** The communication performs well between the RPI and the Teensy. When the delay time was reduced, it would be harder for the camera to get a stable image from the video, but a motion-blurred one. This makes the camera missed many light spots on the ground.

**Corrective actions taken:** Increased the delay time to 0.8 seconds. Increased the video frame to 30.

## Test 3: Spinning and Tracking

Done by Yidi and Christopher

**Test results:** A certain sound has been produced, and the current angle will be recorded when facing a new max. After the spinning has been done, the robot usually undershot compared to its initial heading angle. When the angle of the flashlight changed, the robot is able to track, but unpredictable.

**Comparison:** The new max and the corresponding angle have been well recorded. Tracking is bad.

**Analysis:** The readings from solar panel are reliable. And the Spin function logic is correct. The robot’s undershooting means that the number of turnings we set is not enough, and the motor torques are weak. Either of the tracking logic or the parameters we set cannot fit into the current light conditions.

**Corrective actions taken:** Increased the number of turnings in the spin function. Increased PWM outputs for a higher turning torque. Still cannot navigate bugs in tracking logic.

## Test 4: GSA

Done by James

**Test results:** A 90cm x 90cm grid programmed caused the robot to move roughly 105cm x 110cm. This is an average reading over more than 15 runs.

**Comparison:** Robot moved further than expected and did not check entire grid properly

**Analysis:** Each block of 30cm was set at too many encoder ticks for robot.

**Corrective** actions taken: Ticks per block hard-coded as lower (~40 instead of 48).

## Test 5: Return to Max

Done by James

**Test results:** In 20 runs, over 95% of the time the robot found the bright spot. However, despite finding the spot it did not return to the correct spot.

**Comparison:** Finding of the spot passed the test. Returning to the spot needed to be fixed.

**Analysis:** The angle and distance calculated to return to the max spot must be coded wrong. Debugging proved the distance calculated was faulty and needed recoding.

**Corrective actions taken:** A few lines of code were corrected and now the robot finds and returns to the spot >95% of the time.

# Conclusion and Future Work

## Conclusion

We are happy to say that our project meets all the major project goals that we set out to in winter quarter. Our main objectives were to have a self-charging robot that could position itself in an ideal location. Our plan was to have the GSA operating by the end of fall quarter and the CSA by the end of winter quarter. Our project meets the technical design objectives we set: the robot can detect and move to an optimal bright location at a 95% accuracy rate within a reasonable time frame. The full charge time of the batteries is less than 6 hours using the solar panel alone and can expand the grid size up to a 10ft x 10ft area.

James: Personally, aside from research projects as part of a bigger team, I had not worked on many big projects before. This was a new experience for me and I learned a lot in part selection, schematic/circuit and algorithm development, and planning/time management. There was a lot of freedom in what parts and language to use for the entirety of the design. As a result, we ran into many problems and obstacles in this learning process. It was a good experience in preparation for future professions in this industry. I learned how to perform better research online and how to use the parts we selected through datasheets/code. I also became more familiar with motors, motor encoders, compasses/IMUs, solar panels/photoresistors, how to solder, and many more. It gave me more experience in how to work as a group and communicate with others.

Yidi: It’s my first time to have such a time-consuming project. We did not have an absolute group leader and we help, encourage each other, and this was a very enjoyable experience for me. We followed the schedule to complete the project step by step, although sometimes we fell far from it when encountered a big difficulty. The most impressive thing I have learnt was that, as a EE student, comparing to trying to debug on the software level, solving a problem at the hardware level once for all was more efficient. The project has given me a fully comprehension of the interfaces between software and the hardware, and I got to know that the hardware tests could never be underestimate and the time it would cost is sometimes unpredictable. Anyway, I enjoy the working experience with James and Christopher, and very appreciate their help to me and contributions to our final product, SunE-Bot.

Christopher:

I was very glad to see that we met our major goals by the end of the quarter. We faced some major setbacks with problems with the charging circuit, the encoders, digital compass, and IMU that I did not foresee. Having worked with encoders in the past, albeit two channel magnetic encoders, I thought we could have them working within a week. I did not expect to have a four-week debugging process before we got them working. Similarly, with the digital compass IMU, I completely underestimated how long it would take us to be able to properly interface with them. As a result of these setbacks, movement, which we expected to finish by Week 8 of fall quarter took us until well into winter quarter to finally finish. Despite these challenges, SunE-Bot could accomplish its major goals by the final demo day and I was very proud of that. Overcoming major setbacks such as these taught me to not underestimate that time it would take to accomplish seemingly simple objectives, and skills of debugging and finding new solutions to problems. This project has also taught me how to manage my time and work with a team. I normally try to be involved with every aspect of a project, so I have a complete understanding of as much as I can. In this case, there we major aspects that I entrusted my teammates to accomplish. For example, we had Yidi handle the image processing and James handle the Traversal function which our both very critical to the operation of SunE-Bot. By trusting them to handle these aspects on their own, it gave me time to work on other aspects of the project.

## Future Work

As the cost of the production of solar and robotics technology continuously goes down while simultaneously their effectivity, efficiency, and capabilities grow, we will see an expansion in the field of autonomous robotics. On top of that, for environmental, economic, and public image reasons, businesses are moving towards the use of solar energy. SunE-Bot can be equipped with a variety of additional sensors or actuators and can provide an energy efficient alternative to tedious or costly tasks that do not necessarily require a human to perform.

SunE-Bot is still in its early stages of prototype development, and before we consider its design as a sellable product, there are numerous of improvements we would like to see. For starters we would like to design an automated electronic switching circuit for the power source so that we would not have to manually switch the batteries. Another improvement that could be beneficial, though it will add weight and power consumption, would be an additional motor to adjust the angle of the solar panel. We have it fixed at a 35° angle, which is the optimal angle for our geographical location, but it would be more effective if we could adjust it could more precisely track the sun. There are many improvements we would like to implement for more precise movement. We would like to switch to higher quality motors that could operate at lower speeds (possibly stepper motors for further accuracy), encoders on each wheel for better data, and a better tire that provides more grip to the ground. We would like to add some form of obstacle avoidance to the SunE-Bot, through use of sensors, computer vision, or both. Since early in the design stage, we were also hoping to see an LCD interface that could improve the user interface. We would also like to see improvements in the SunE-Bots aesthetic by upgrading the body to something with a cleaner look. If we would also like to implement our software using ROS as it can make our program more portable to other robotics projects. Finally, we would always like to continue improving the power efficiency of the SunE-Bot because its main purpose is to be as energy efficient as possible. We could modify the charging circuit, so power does not get wasted when the battery is fully charged and make a more efficient charge controlling circuit that does not need to step down the voltage coming in from, the solar panel. We could also try to find more power efficient parts.

## Acknowledgement

We had a lot of help from the people and resources on campus at the University of California, Riverside with building the SunE-Bot. We would like to thank the following people and organizations.

1. **Konstantinos Karydis:** Professor Karydis provided guidance and helpful suggestions throughout the course of the project. His leadership helped us keep the production of our project on track.
2. **Krishna Addanki:** Krisha is a fellow student who gave us the suggestion of switching our batteries from AA NiMH batteries.
3. **Manglai Zhou:** Mr. Zhou graciously allowed us to use soldering equipment and other tools in his lab as well as access to his 3D printer for our solar panel holder.
4. **IEEE:** IEEE is the Institute of Electrical and Electronics Engineers and they have an organization on campus which allowed us to borrow some parts. They also provided 3D printing for some of our LED holders.

We had technical help from online resources and creators who provided useful guides and open source code for our robot’s sensors.

1. **Bill Earl:** Although we did not end up using the compass in the final version of the SunE-Bot, his

guide on the LSM303 digital compass and open source code was helpful for our testing and experimentation of the compass.

link: [*https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout?view=all*](https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout?view=all)

1. **Aritro Mukherjee:** His guide and open source code for using the MPU 6050 was used in testing

our IMU.

link: *https://create.arduino.cc/projecthub/Aritro/getting-started-with-imu-6-dof-motion-sensor-96e066*

1. **K. K. Poon:** He provided open source code for the calibration of the MPU 6050.

link: *https://github.com/kkpoon/CalibrateMPU6050*

# References

Arduino Project Hub. (2018). Getting started with IMU (6 DOF) motion sensor. [online] Available at: https://create.arduino.cc/projecthub/Aritro/getting-started-with-imu-6-dof-motion-sensor-96e066 [Accessed 15 Feb. 2018].

cdn.sparkfun.com. (2018). [online] Available at: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/K66P144M180SF5RMV2.pdf [Accessed 11 Oct. 2018].

dlnmh9ip6v2uc.cloudfront.net. (2018). [online] Available at: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf [Accessed 27 Oct. 2018].

GitHub. (2018). kkpoon/CalibrateMPU6050. [online] Available at: https://github.com/kkpoon/CalibrateMPU6050 [Accessed 14 Feb. 2018].

Instructables.com. (2018). Mobile Sun-Seeking Robot. [online] Available at: http://www.instructables.com/id/Mobile-Sun-Seeking-Robot/ [Accessed 4 Oct. 2018].

learn.adafruit.com. (2018). Overview | LSM303 Accelerometer + Compass Breakout | Adafruit Learning System. [online] Available at: https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout?view=all [Accessed 1 Feb. 2018].

Sparkfun.com. (2018). [online] Available at: https://www.sparkfun.com/datasheets/Robotics/L298\_H\_Bridge.pdf [Accessed 21 Oct. 2018].

Store.invensense.com. (2018). [online] Available at: https://store.invensense.com/datasheets/invensense/MPU-6050\_DataSheet\_V3%204.pdf [Accessed 14 Feb. 2018].

Raspberrypi.org. (2018). [online] Available at: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1\_0.pdf [Accessed 15 Jan. 2018].

# Appendices

Presents information that supplements the design specification, including:

**Appendix A:** Parts List

|  |  |  |
| --- | --- | --- |
| **Quantity** | **Parts** | **Details** |
| 1 | Raspberry Pi 3 Model B | including holder, heating sink and charging cable |
| 1 | Teensy 3.6 |  |
| 1 | Raspberry Pi Camera Module |  |
| 1 | Solar Panel | 12V, 5W |
| 4 | 18650 Battery Set | 3 used in prototype |
| 1 | IMU | MPU 6050 |
| 2 | Optical Encoders | HC-020K |
| 1 | Car Chassis | With wheels and tires |
| 4 | DC Motors | CYT1037 |
| 2 | Motor Drivers | L298N Dual H Bridge |
| 1 | Camera Case |  |
| 1 | Charge Controller | TP-4056 |
| 1 | Step-Down Converter | UBEC DC/DC 5V |
| 3 | Battery Holders | 18650 Size |
| 4 | Photoresistors |  |
| 1 | Schottky Diode | 1N5820 |
| 2 | Ceramic Capacitors | 104 capacitors |
| 2 | Through Hole Board |  |
| 1 | Breadboard |  |
| 4 | Custom Printed Photoresistor Holders |  |
| 2 | Custom Printed Solar Panel Holder |  |
| 2 | Switches |  |
| 1 | Buzzer | KPI1410 |
|  | Wire | solid |
|  | Jumper Cable |  |

**Appendix B:** Equipment List

|  |  |
| --- | --- |
| **Item Description** | **Location** |
| 3D printer | WCH 137 |
| Hot glue | WCH 137 |
| Soldering kit | WCH 137 |
| DC Power Supply | WCH 126 |
| Oscilloscope | WCH 126 |
| Multimeter | WCH 126 |

**Appendix C:** Software List (URL to online drive or SVN server, with sharing set to Public. Can omit this appendix if your project didn’t involving writing a program)

|  |  |
| --- | --- |
|  | URL |
| Code | https://github.com/cverd001/Senior-Design |
| Drive | https://drive.google.com/open?id=0Byn20I8b1E-sZF9DRGFsUnV0Vm8 |

**Appendix D:** User Manual

Turn on SunE-Bot by activating the two switches towards the user. Menu will begin. Refer to Section 8.2 for user interface information. After menu selection, place down SunE-Bot in a room with a smooth surface and it will operate autonomously. When done, turn off the switches.