Advanced Laboratory: Developing Skooter and Espy

A Project by Erik Brobyn, Alexandros Gloor, and Noah Hovde

Chapter I - Preparation and Calibration

Ia. General Circuitry and Setup

The assembly of Skooter was a tricky process. The entire machine, from tracks to LIDAR, had to be disassembled and reassembled several times. After countless implementations of the robot's setup, we settled on a design that allowed for enough empty space to house all the components, while maintaining stability and integrity long enough to last through the demonstration and subsequent activities. In particular, toward the conclusion of the project, we decided to modify the chassis in order to securely fasten the pan-tilt configuration which required a complete reassembly. Noah assisted, mainly, in the disassembly and reassembly of Skooter, but also with the measurements required to mount the pan-tilt configuration onto the chassis. The wiring was originally done by Alexandros. However, later on in the process, a breakout implementation of just the SD card, Arduino, and Espy was also required for unit-testing and debugging purposes--Erik ultimately wired the final build.

Ib. Independent Operation of Step Servos

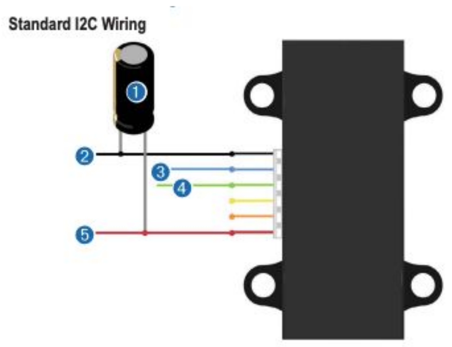
The two step servos that support Skooter's eyes (LIDARLite), which we often refer to as the "Tilt and Pan" servos, were, initially, put together by Alexandros. However, there was a problem with the stability of the step servo controlling the tilt angle. This required essentially a full disassembly of the pan-tilt configuration, performed by Erik and Noah, down to the properly functioning step servo controlling the pan angle. From there, Noah rebuilt the pan-tilt configuration and in the process restricted the tilt angle so as to avoid tangling the wiring. However, upon closer inspection, it was decided that a wider tilt angle would be necessary for the sake of our project, so Noah disassembled pan-tilt configuration and altered a few of the components to give a wider range to the step servo controlling the pan angle. We also had some issues with screws falling off the assembly. These servos worked perfectly out of the box off using the Arduino servo library, and wiring was straight forward (one power, ground, and analog pin for each servo), using I2C.

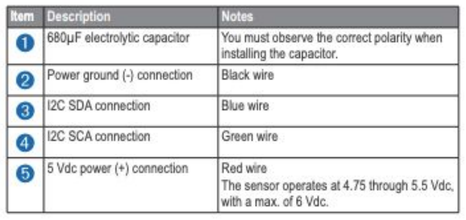
Ic. Independent Operation of Continuous Servos

Alexandros and Erik used considerable trial-and-error with different modes and dip-switch configurations. Alexandros originally took up the task of getting these operational with their own library. While they ran, they would not do so continuously, often behaving randomly, meaning troubleshooting had to be conducted. The Sabertooth 2x5 RC dip-switch settings were difficult for us to nail down. We found the manual provided with the RC servos lacking and difficult to troubleshoot.

Once the correct library was used, the behavior of Skooter’s tracks was tested. We discovered that the settings used on the servo driver had to be changed to continuous, independent mode, without auto calibration on setup.

Id. Independent Operation of LIDAR-Lite v3





Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

Setting up the LIDAR-Lite required the installation of a new library and the careful reading of the data sheet provided by the manufacturer. Wiring and coding implementation was done by Alexandros. This was complicated and required the addition of a capacitor of a specific capacitance and while work was put in using the already available capacitors in series and parallel to get the desired capacitance (680µF), a capacitor of this specific value was found and used instead. This circuit had its power and ground placed in parallel to the rest of the circuitry with no unwanted side effects.

In order to get the custom library functional, various setups were tested, finally settling with I2C mode. The LIDAR-Lite was operated as a slave device in the I2C mode. This made operation particularly easy, but caused issues with the delay() function which are discussed in section IVb. Regardless, the LIDAR now operate well as long as the delay() functions were isolated into the servo file alone.

Ie. Powering Skooter

The power bank that came with Skooter’s chassis powered the motor-driver perfectly, and at first it also fed power to the arduino board and every other system in the circuit. While conflicts were theorized at first, no issues were found with running Skooter’s main chassis operations. Ultimately however, we did require an additional power source of 5V for the Arduino in order to run the SD card and ESP8266-01 along with all the other servo and sensor components.

If. Independent Operation of the SD Card

Initial investigations into file IO operations on the SD card were done by Noah. He began with the SD ReadWrite sample sketch provided via the Arduino IDE. From there, the SD card was wired to the proper pins on the Uno and the chip select pin within the code was switched to the according pin on our microcontroller. Subsequently, the code was refined and altered in order to write to and read from the SD card files formatted as comma separated variables, specifically Strings.

However, since the Cabinet class also handles Espy requests, it became clear that a state machine implementation would be required. When communicating to Espy via the SoftwareSerial instance and its tiny buffer, blocking calls must be completely eliminated. For this reason, extensive modifications were made, by Erik with assistance from Noah, to remove delay() calls during file and directory open and close calls and other critical operations on the SD card during Skooter's main loop routine.

Also, it was observed that not all files are deleted or removed when those calls are attempted (there appears to be some flash or other cached data from within the SD card itself that the API calls do not affect). Similarly, sometimes calls to open a file can fail, and when this occurs it is sometimes necessary to completely reset the SD instance.

For the most part however, the SD card functions work great, and we were able to save data files during scan routines of varying size without issue. Data requests (via the SoftwareSerial interface) are generally handled robustly from this side without incident.

Ig. Independent Operation of ESP8266-01

Espy, or the ESP8266-01, was set up by Erik and Noah. Noah provided the necessary research and background information pertinent the internet chip, ESP8266-01, and the implementation of a web server.

In addition to its wifi client, Espy also has an internal 1M flash, of which up to 512K can be used for a file system. Given this feature, an important milestone of serving larger files (<<512k) became feasible through the use of a simple caching algorithm. In this scenario, Espy would simply forward the request to Skooter and return a simple message back to the original client, letting them know the request is being processed. Meanwhile, the requested data is then cached into the SPIFFs (SPI Flash File system) on board Espy. On subsequent requests, the file is then found locally, and can be streamed quickly to the next client.

Ih. Implementation of the Piezo-Buzzer Class

Erik implemented the Noisemaker class to provide auditory feedback to the user. Noisemaker has a startup noise (3 beeps) and another "I see you!" noise. Noisemaker is the one class in Skooter that still uses delay() as of this writing. Due to its extensive use, all other activity is therefore blocked while Skooter is making a noise.

Chapter II - Skooter

IIa. Classes

Skooter's project contains a number of lower level classes which are used mainly by ***Skooter** class implementations.

Tracks - An interface between the Sabertooth 2x5 RC motor driver and the Arduino. An instance of Tracks holds private references to 2 servos, a left and a right. Implementations for goForward(units), turnLeft(degrees), turnRight(degrees), stop() etc.

Cabinet - An interface between the Arduino and a micro SD card reader. Public methods include writeToCurrentFile(line), writeCurrentFileName() etc.

PanTilt - An interface between the Arduino and the Pan-tilt servo assembly. A PanTilt instance holds private references to a pan servo and a tilt servo. The entire assembly gives Skooter's lidar sensor the ability to tilt approximately 90 degrees and pan approximately 180 degrees about its 0 degree heading (facing forward, parallel with the ground). Public methods include pan(degrees), tilt(degrees), etc.

Noisemaker - A simple piezo-buzzer wrapper.

Lidar and **LidarData** - A simple wrapper for accessing the LidarLITE api, and a data structure that encapsulates a complete lidar reading with position and angle orientations, respectively.

High-level classes include **Skooter**, **MotorSkooter**, **etc** - These *Skooter classes are essentially high-level classes containing various experiments, unit-tests, and algorithms that determine Skooter's behavior. They all contain loop() and setup() methods, and are probably best implemented as state machines with their own high-level state transitions for best performance and data-integrity.

IIb. Repository and Milestones

Most interactions with the repository (uploading and updating) were done by Erik as his laptop was our primary engine. This was done due to the ease of using Visual Studio on Windows and the fact that neither Noah nor Alexandros could run Windows on their laptops. Notable commit dates in the repository are listed below:

- Nov 15-16 - Initial project code-base checked into repository created (Erik)
    - MILESTONE: Repository Created
- Nov 20 - Cabinet IO, SD card interface sketched out (Noah)
    - MILESTONE: Simple file IO unit-tests
- Nov 20 - LIDAR (Alexandros)
    - MILESTONE: Lidar unit-tests
- Nov 23 - Tilt and Pan proof-of-concept working (Alexandros)
    - MILESTONE: Pan Tilt unit-tests
- Nov 24 - Refactoring, comments, and clean-up (Erik, Alexandros, Noah)
- Nov 29 - Tracks - initial commit (Erik)
- Nov 29 - MILESTONE: Tracks (motor driver) unit-tests (Alexandros)
- Dec 8 - Noisemaker (piezo-buzzer) implemented (Erik)
- Dec 16 - Skooter's "scan" algorithm
    implemented (Erik, Alexandros)
    - MILESTONE: Scan algorithm implemented, tested and data integrity verified.
      The scan algorithm, in pseudocode, is: "while stationary, perform a raster scan using the full range of pan and tilt motions, by some fixed angle increment, and take a lidar reading at each point. Save all readings to file"
    - MILESTONE: File served by Espy in its entirety
- Dec 16 - MotorSkooter class (simple path-finding algorithm without delay())
    - MILESTONE: MotorSkooter: Simple Path-finding algorithm tested

IIc. Interaction Between LIDAR-Lite and Step Servos

Alexandros developed multiple programs that implemented the Tilt and Pan step servos to provide meaningful, extractable data to Skooter. These included:

- High accuracy solid angle measurement of distances.
- Fast look mode that checked the distance in front of the robot as it was moving.
- Three-point distance measurement for naive AI motion.
- 3D (multiple position solid angle) mapping of point-of-interest.

A majority of these programs did not end up getting used in the final program due to their heavy dependence on the delay function, but were reimplemented and redesigned into the main operation of Skooter that was seen in the final project.

IId. Main Chassis

In order to assure that the LIDAR-Lite was always facing forward and that the motion of Skooter and his step servos would not cause the whole LIDAR/Servo section to fall apart, new holes had to be drilled into the chassis of Skooter. Measurement of the required drill size, positioning, alignment, and orientation was done by Noah, Alexandros, and Erik. Drilling was done in the lab using a hand tool. In the end, this allowed us to replace the tape that we originally used to hold the LIDAR down with actual screws that was significantly more reliable and sturdy during regular drive.

IIe. Motion Detection

Erik implemented a simple motion detection method to Skooter that checks whether distances between two measurements done by the LIDAR-Lite at a single position varied by a significant amount. Triggering the motion detector was coded to set the buzzer off and mark it as a point of interest.

IIf. Naive AI

The final product used significantly more naive artificial intelligence than originally planned. Designed by Alexandros and coded by Erik, this had to be limited due to timing conflicts and our desire to implement Espy and have communication between the files and server functioning correctly. The biggest deterrent to our implementation to a better AI was our inability to use the delay() function (discussed in section IVb), which required us to shift to a state machine.

Chapter III - Espy

IIIa. About ESP8266-01

Espy is an ESP8266-01 wifi client that can be converted into a web server using an out-of-the-box example available in the Generic ESP8266 board extension within the Arduino IDE. The web server example code had to be modified to support serial buffering to and from the

Arduino. It was Espy that gave us the initial clue that we really needed a state machine solution throughout Skooter's functionality, since http transactions are sensitive to time-outs. Espy is a "pure" state machine, in the sense that it is completely free of any blocking calls.

IIIb. Classes

Espy's behavior is mostly defined in the EspySketch.ino file, but there are also 3 static helper classes:

EspySerial - State machine implementation to read/write to the serial buffer
EspyRequestHandler - Format strings to return as responses to http requests
SkooterFiles - A data-structure for containing a list of Skooter's current files

IIIc. Repository and Milestones

Interaction dates to the repository are listed below:

Dec 2-4 - Initial investigation of the ESP8266-01 (Noah)
Dec 6 - Inter-process communication between Arduino and ESP8266-01 (Erik)
Dec 7 - Espy web-server implemented (Erik)
MILESTONE: Espy appends new file name to the links on the main page whenever Skooter creates a new file
MILESTONE: Espy requests file contents from Skooter and caches the results locally for subsequent follow-up http requests

Chapter IV - Troubleshooting / Lessons Learned

IVa. C++ VS C

Due to the limited memory bandwidth on Arduino microcontrollers, we found it preferable to work in C, especially for core implementation or memory-critical operations, while still maximizing utility of C++ for naming, higher-level functions and/or concepts. In particular, we came across memory issues with the String implementation in Arduino. We favor char buffers and fixed size arrays over Strings for any memory-critical or data-intensive operations, and favor minimizing the number of temporary and static variables, and static class instances whenever possible.

IVb. The delay() Function

The delay() function, although helpful for illustrating functionality of individual components or for example code, is problematic for complex systems that require continuous feedback or data. In particular, inter-process communication is critically negatively affected by using delay(), since serial reads and writes are done using a buffer of limited size, and buffer use must be synchronized between 2 separate processes (one running on the Arduino, the other running on the ESP8266-01). However, even calling delay() while doing two seemingly isolated and independent activities, for example, writing data to a file while also speeding up a servo, could also cause considerable error-prone behavior and/or failures. For this reason, a major portion of the development effort went towards removing those calls from our code-base after some simple (but nevertheless broken) algorithms had already been implemented.

IVc. Implementation of the State Machine:

A process is a thread of activity that is continuously running in a loop or while routine. Examples are a web server, an operating system, or the execution of an Arduino or ESP8266-01 loop() method during runtime. This process may be implemented as a state machine if it possesses some state which determines the current action taken. This state is then potentially updated during each iteration of the loop. Rather than block between calls that turn on and then off some functionality using delay() in between, the process' state can instead simply be changed to reflect that calls to turn on or off the desired behavior have been made. In this way, no blocking calls are required to achieve the same effect, and the process can perform other actions during those loop iterations that would not otherwise get called during a blocking call.

IVd. Servo Inconsistency

Skooter's Track servos operate at a different speed, depending on whether they are moving forward or backward, so for example +40 will not be the same speed as -40. In general, we found a ratio of about 4:5, in favor of forward motion. This impacted our turn algorithms because we had to ensure the backward moving servo kept up with the forward moving servo. Although a number of potentially more robust solutions were proposed, we settled on hard-coded values, assuming a fully charged battery, due to time constraints. Algorithms that were only partially implemented due to calibration limitations include turning to a specific angle, making a complete circle etc.

IVe. Source Control & C++ Language Concepts

With the implementation of the Github repository at the very beginning of this project, members of our team unfamiliar with such practices were faced with a steep learning curve. Noah, for example, had to learn the entire system of pushing and pulling edits to and from the repository, how to save work to local directories to ensure nothing was lost or overwritten, etcetera. In doing so, he learned about concepts such as source control and how to effectively stay on top of and manage changes to the code-base.

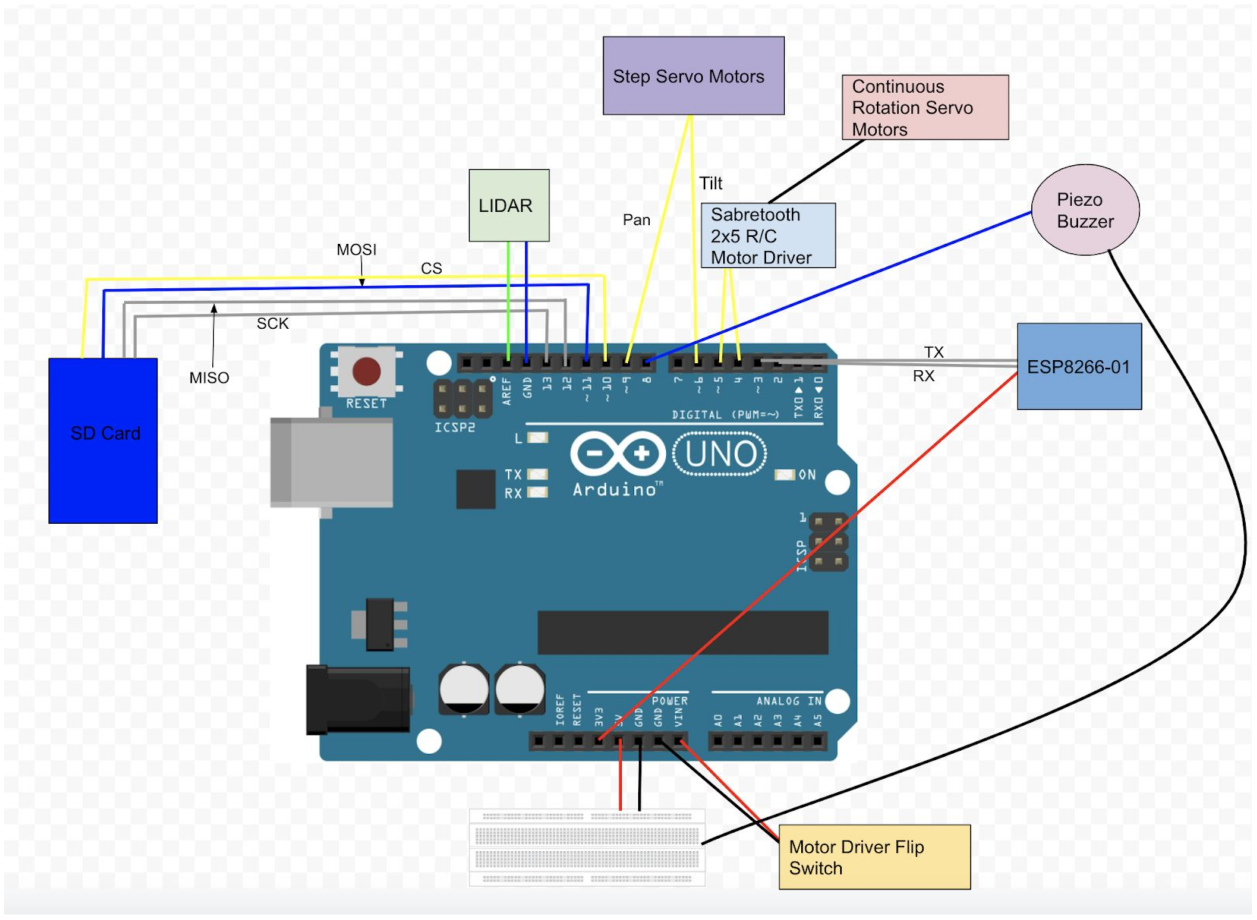
Noah managed to gain experience in C++ and Object Oriented Programming concepts, and learned about concepts such as scope rules, static versus local variables, preprocessor directives (#define, #include), class methods, access specifiers (public vs private membership), class constructors and destructors, and enumerations. Noah learned about creating function prototypes (contracts) in header files and also wrote implementations (definitions) for several methods. Inheritance concepts were also discussed and explored. Noah also learned about external libraries, specifically how to write one and how to include it into a code-base.

Chapter V - Future

Skooter has multiple fascinating potential functionalities. We see using Skooter as a generic test-bed for SLAM, computer vision and motion tracking, and spatial mapping algorithms. By improving the AI that it runs on, Skooter can truly perform a full room 3D floormap scan using our SLAM algorithm. The implementation of such programs have myriad uses: for example, they could be used as a rescue droid in wreckages and disaster sites that human involvement would otherwise be not recommended. Furthermore, additional software could use Skooter's data to create 3D images of objects. By running around objects and getting accurate point-maps of solid angles, Skooter could be used for 3D printing of scaled down objects.

We'd like to investigate a Unity3D-based mobile application proof-of-concept that reads and creates a point-cloud rendering of the data scans served by Espy. We'd also like to investigate a "follow-the-finger" algorithm (mesmerized mode) to improve Skooter's motion tracking.

Schematic of Circuit



Schematic of Skooter's Circuit--This diagram is a rough depiction of Skooter's circuit and includes all of its main components. Specifically, the SD card module, the LIDAR, the step servos, the continuous rotation servos, the Sabretooth 2x5 R/C motor driver, the Piezzo buzzer, the motor driver flip switch, the breadboard (not in detail), and the ESP8266-01 internet chip.