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

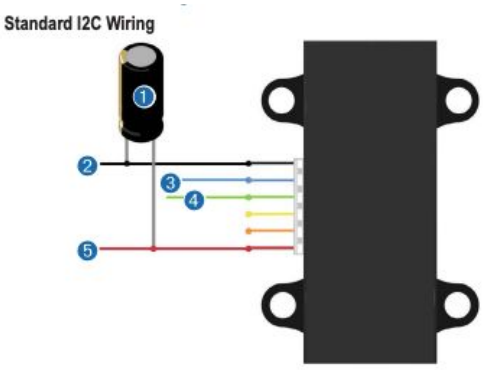
The assembly of Skooter was an arduous process. The entire machine, from tracks to LIDAR, had to be disassembled and reassembled multiple times. After multiple reimplementations of the robot's setup, we settled on a design that allowed for enough empty space to house all the components, and that kept everything snug enough to last through the demonstration and subsequent activities. In particular, we decided to modify the chassis to securely fasten the pan-tilt assembly. While the wiring was original done by Alexandros, the final build required the implementation of the SD card and Espy, which required Erik to rewire the whole circuit.

Ib. Independent Operation of Step Servos

The two step servos that support Skooter's eyes (LIDARLite), often called "Tilt and Pan" servos, were put together by Alexandros. The first implementation of these servos was not perfect so Erik and Noah disassembled and shifted the servos around to give a wider range to the second servo. When moving fast these servos were found to vibrate a lot, which led to screws falling off. This was taken into account in further programming. These servos worked perfectly out of the box off of the Arduino servo library, and wiring was straight forward (one power, ground, and analog pin for each servo), using I2C.

Ic. Independent Operation of LIDAR-Lite v3

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 very specific capacitance and while work was put in using the already available capacitors in series and in 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.



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.

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 IVa. Regardless, the LIDAR now operate well as long as the delay() functions were isolated into the servo file alone.

Id. Independent Operation of Continuous Servos

Operating the continuous servos was particularly tedious. Alexandros and Erik spent countless hours and several evenings playing around with different modes and different pin configurations. Due to our not having access to the packaging that the servos originally came in we had no idea what the actual part number was. Alexandros originally took on the job 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. After a few more nights of working on this with Alexandros, Erik discovered that these were RC versions of the servos which Alexandros originally thought he was working with and the library was replaced with the correct RC one. The manual provided with the RC servos was also significantly less detailed which required much more trial and error.

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. A baud rate of 9600 was also important to include. Aside from that no issues were encountered.

Ie. Powering Skooter

The power bank that came with Skooter's chassis powered the servo driver perfectly, which 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 a separate 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

The cabinet file and saving data to the SD card was done by Noah.

Ig. Independent Operation of ESP8266-01

Espy, or ESP8266-01, was set up by Erik and Noah.

Ih. Implementation of the Piezo-Buzzer

Was done by Erik to provide auditory feedback to the user.

Chapter II - Skooter

IIa. Classes

Skooter's project contains the following classes which are used repeatedly in the project.

Tracks - Interface between the Sabertooth 2x5 RC motor driver and the Arduino.
Cabinet - Interface between the Arduino and a micro SD card reader.
PanTilt - Interface between the Arduino and the Pan-tilt servo assembly that provides Skooter's lidar sensor with the ability to tilt approximately 90 degrees and pan approximately 180 degrees.
Noisemaker - Simple piezo-buzzer wrapper.
Lidar and LidarData - Wrapper for accessing the LidarLITE api and wrapper struct for encapsulating readings.
Skooter, MotorSkooter, etc - High-level classes containing various experiments, unit-tests, and algorithms that determine Skooter's behavior.

IIb. Repository

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. Interaction dates to the repository are listed below:

Nov 15-16 - Initial project code-base checked into repository created (Erik)
Nov 20 - Cabinet IO, SD card interface sketched out (Noah)
Nov 23 - Tilt and Pan proof-of-concept working (Alexandros)
Nov 23 - LIDAR (Alexandros)
Nov 24 - Refactoring, comments, and clean-up (Erik & Noah)
Nov 29 - Tracks - initial commit (Erik)
Nov 29 - Tracks Unit-Tests (Alexandros)

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 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 checked whether the distance between two measurements done by the LIDAR-Lite at a single position varied by a significant, modifiable number of centimeters. 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 communications with the server functioning correctly. The biggest deterrent to our implementation to a better AI was our inability to use the delay() function (discussed in section IVa), which required us to shift to a state machine.

IIg. State Machine

The eventual state machine is discussed in further detail in section IVa. This state machine was then implemented by Erik for most of the problematic operations with the exception of the buzzer that was edited by Alexandros.

Chapter III - Espy

IIIa. Classes

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

EspySerial - State machine 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

IIIb. Repository

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)
Dec 8 - Noisemaker (piezo-buzzer) implemented (Erik)
Dec 16 - MotorSkooter class (simple path-finding algorithm without delay()) implemented (Erik, Alexandros, Noah)

Chapter IV - Troubleshooting

IVa. The delay() Function

The delay() function, although helpful for illustrating functionality of individual components or example code, is problematic for complex systems that require continuous feedback or data. In particular, inter-process communication is negatively affected by using delay(), since serial reads and writes are done using a buffer of limited size, and this 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 activities, for example, writing data to a file while also revving a servo, could also cause failures. For this reason, a major portion of the development effort went towards removing those calls from our code-base after some simple algorithms had already been implemented.

Implementation of the State Machine:

A process is a thread of activity that is continuously running in a loop or while routine. Examples are the execution of Arduino and ESP8266-01 loop() methods 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(), the process' state is instead simply changed to reflect that calls to turn on or off 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.

IVb. Continuous 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.

Chapter V - Future

Skooter has a lot of potential and multiple functionalities. By improving the AI that it runs on, Skooter can truly perform a full room 3D floormap scan using our SLAM algorithm. Not only that, but the implementation of such programs as the point of interest motion detection would allow Skooter to be used as a rescue droid in wreckages and disaster sites that human involvement would otherwise be not recommended. Furthermore, Skooter is able to create 3D

images of motionless objects by running around them and getting accurate point-maps of solid angles, allowing it to be used for 3D printing of scaled down objects. A Unity3D based mobile application proof-of-concept that reads and creates a point-cloud rendering of the data scans served by Espy will be investigated. Follow-the-finger (Mesmerized mode) will be investigated for improved motion tracking