

- Project 1: Catme survey

Project 2 Lessons

Project 2 Lessons

Functions can be abstractions

- Hide details from their “callers”
- In our case: we are hiding the details of the analog interface and how to interpret the analog values
- Functions should adhere to their specification and do no more

Project 3: Lateral Velocity Sensing

Project 3: Lateral Velocity Sensing

Each hovercraft has 3 downward-looking cameras at different angles

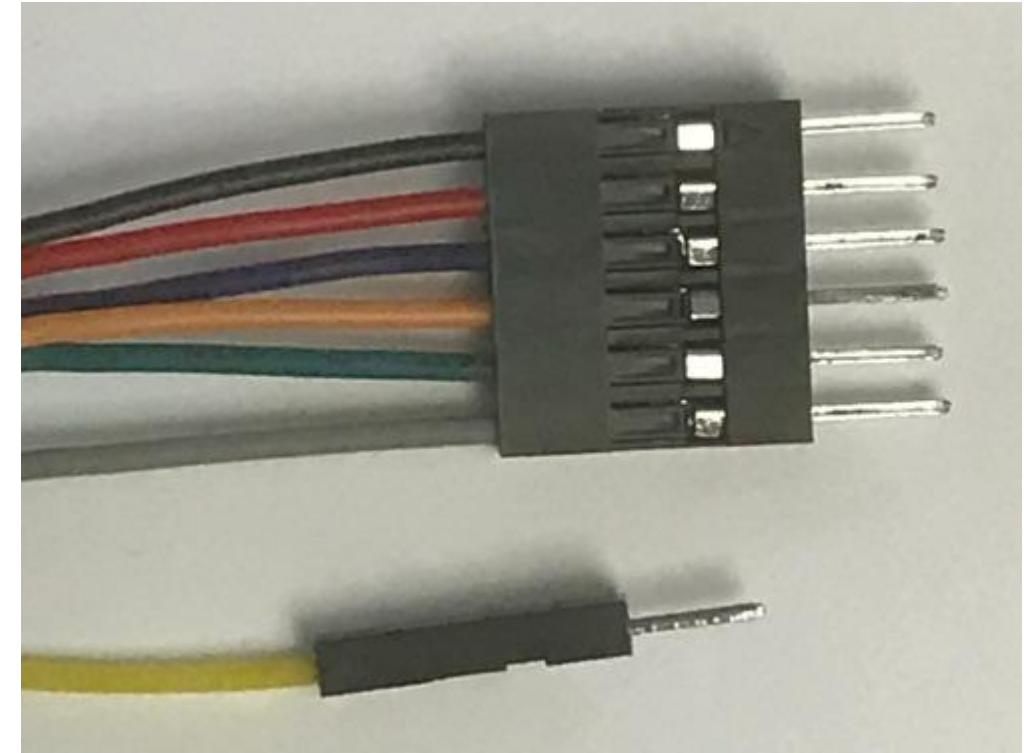
- Connect to a Serial Peripheral Interface (SPI)
 - High-speed serial bus
- When you query a camera, it will tell you how many pixels of “slip” have happened since the last time you asked
 - Both X and Y components
- With two or more cameras, we can estimate how far the craft has moved in three dimensions

Component 1: Physical Interface

Common across all cameras:

- Black: Ground
- Red: +5V Power
- Blue: MISO (Arduino pin 12)
- Orange: MOSI (Arduino pin 11)
- Green: SCL (Arduino pin 13)
- Gray: Reset (choose an unused digital pin)

Each camera has a yellow select line (choose a unique, unused digital pin)



Component 2: Supporting Types/Implementation

Top of program:

```
// Promise that we will implement this function later void void
void camera_step();
void report_step();

// Create a task that will be executed once per 50 ms
PeriodicAction camera_task(50, camera_step);
PeriodicAction report_task(1000, report_step);
```

Loop:

```
void loop()
{
    // Check to see if it is time to execute the tasks
    camera_task.step();
    report_task.step();
}
```

Global Variables

```
int32_t adx[3] = {0, 0, 0};  
int32_t ady[3] = {0, 0, 0};  
float cartesian_position[3] = {0.0, 0.0, 0.0};
```

camera_step()

- This function is guaranteed to be called once per 50 ms
- Query each camera for how much it has “slipped” since the last query
- Add this slip to adx/ady

Camera Interface

Top of program (example definitions):

```
// Global constants
// Total number of cameras
const int NUM_CAMERAS = 3;

// Select pins for the 3 cameras
const uint8_t CAMERA_SELECT[NUM_CAMERAS] = {8, 7, 10};

// Common reset pin
const uint8_t RESET_PIN = 9;

// Camera interface object
OpticalFlowCamera cameras(RESET_PIN);
```

Reading Slip from One Camera

```
int8_t dx, dy;  
uint8_t quality;  
int result;  
  
// For the ith camera:  
result = cameras.readSlip(CAMERA_SELECT[i],  
                           dx, dy, quality);
```

New behavior in C++ (not seen in C):

- `readSlip()` will change the value of the variables `dx`, `dy` and `quality`

readSlip

```
result = cameras.readSlip(CAMERA_SELECT[i],  
                           dx, dy, quality);
```

- **If result == 0:**
 - dx, dy and quality variables have been changed and can be used
- **If result == -1:**
 - readSlip() is not being called quickly enough
- **If result == -2:**
 - No slip has occurred; do not use dx, dy and quality

Component 4: Data Collection

- Record 10 repetitions of the accumulated values for three types of movement: forward 1m, leftward 1m, rotate clockwise 360 degrees
- Store in a table: a total of 30 rows

Component 4: Sensor Model

- We are estimating the parameters of functions of the forms of:

$$\begin{aligned} x = & a_1 * \text{adx1} + a_2 * \text{ady1} + a_3 * \text{adx2} \\ & + a_4 * \text{ady2} + a_5 * \text{adx3} + a_6 * \text{ady3} \end{aligned}$$

- where $a_1 \dots a_6$ are the coefficients of our function, and $\text{adx?}/\text{ady?}$ are the accumulated slip values

Sensor Model

Use “Multi-Regression” to compute the parameters

- Handle X, Y and theta separately (one set of parameters for each)
- Use all 30 data points to fit each of the three parameter sets

Component 5: Implement the Model

Implement the function:

```
void compute_chassis_motion(int32_t adx[3], int32_t ady[3],  
                           float[3] cartesian_position);
```

Translate adx and ady into hovercraft motion

- Inputs: adx, ady
- Output: cartesian_position

Component 6: Testing

`camera_step()` changes:

- Once per second: compute and print motion

Take five more samples of each motion type: move forward 1m, move left 1m and turn 360 degrees

- For each of X, Y and theta (separately): plot mean and variance for each motion type. A bar graph is good here (a box plot is even better)

Hints

- Start this project early
- Keep things simple