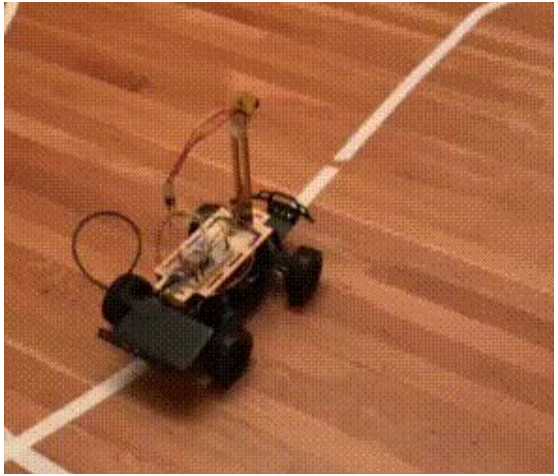
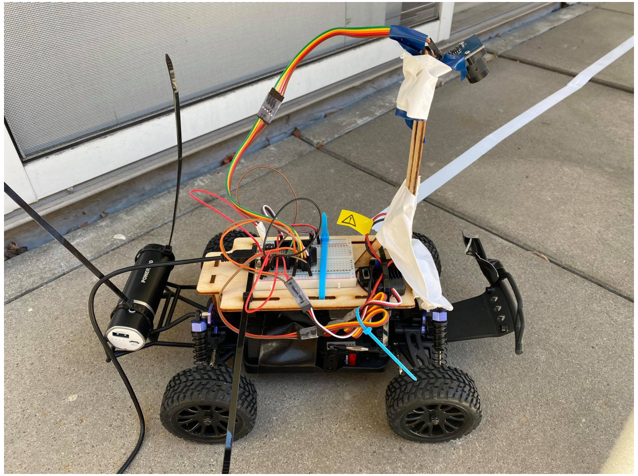
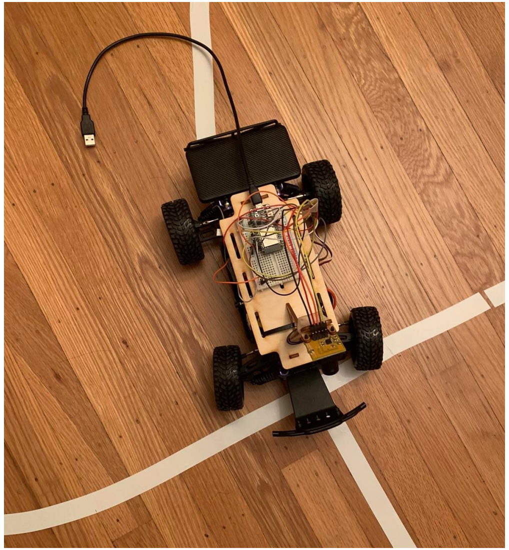
EECS192 Oral Report







Thiti Khomin
Nareauphol Liu
guinea wheek

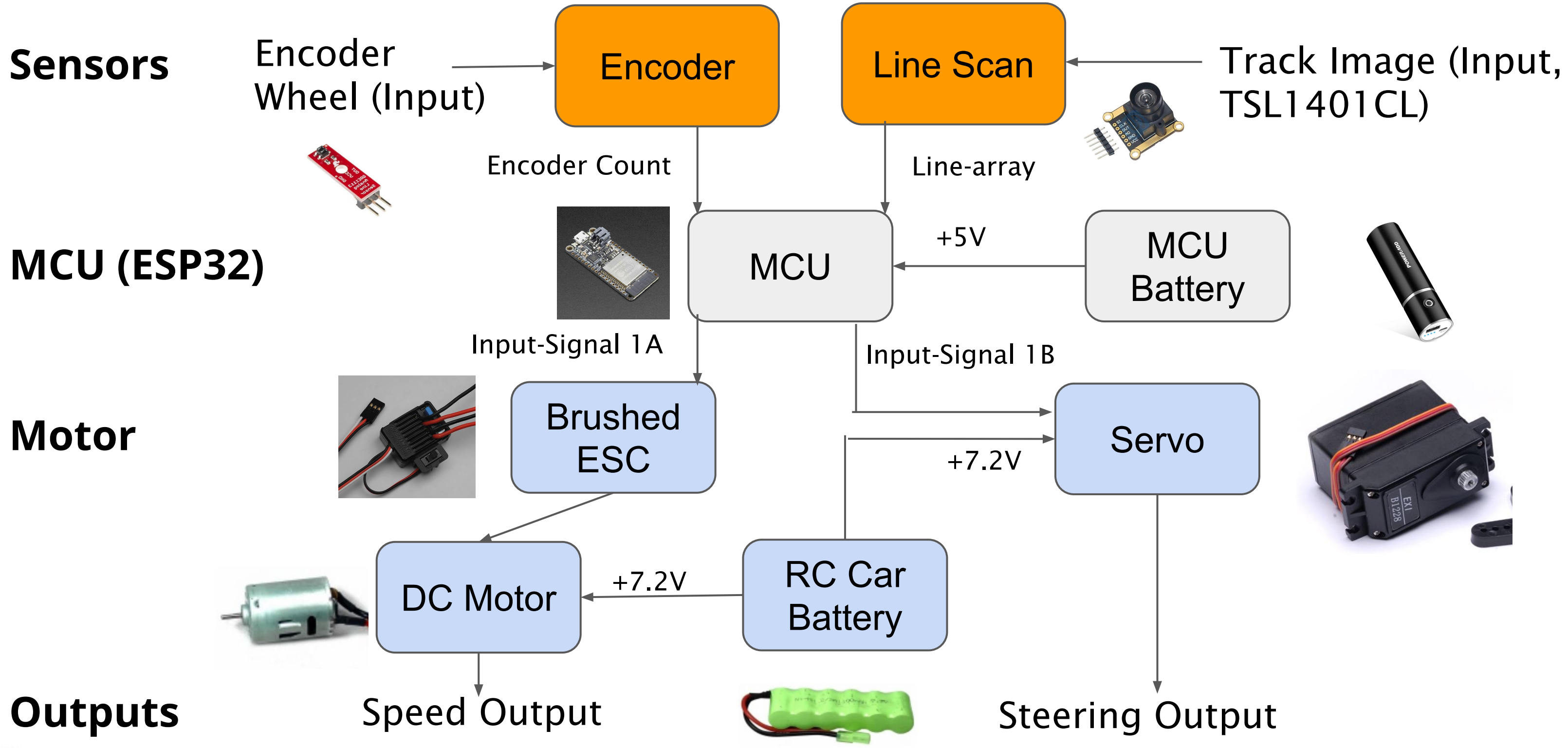
Presentation Overview

Outline

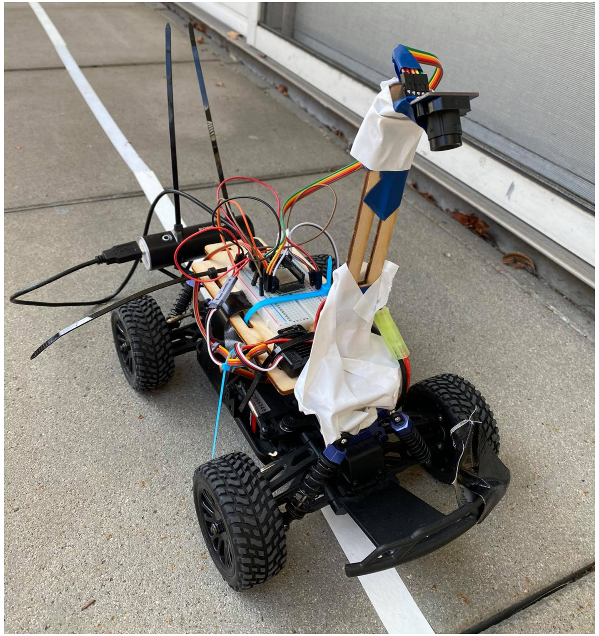
- Project overview
- Vehicle Hardware and Embedded Peripherals
- Line Sensor Algorithm
- Software
- Controls
- Lessons learned
- Roles and Contributions

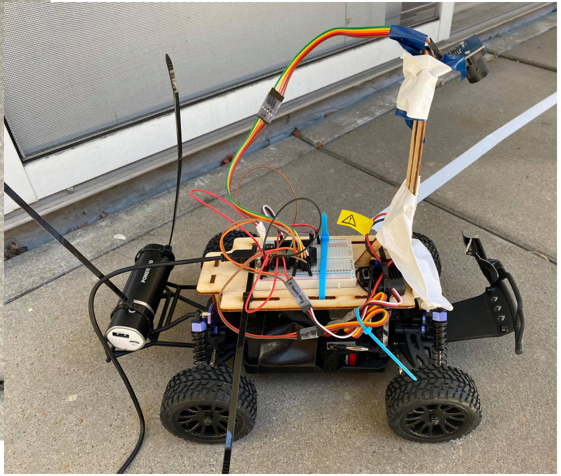
Berkeley
UNIVERSITY OF CALIFORNIA

Hardware Block Diagram/Overview



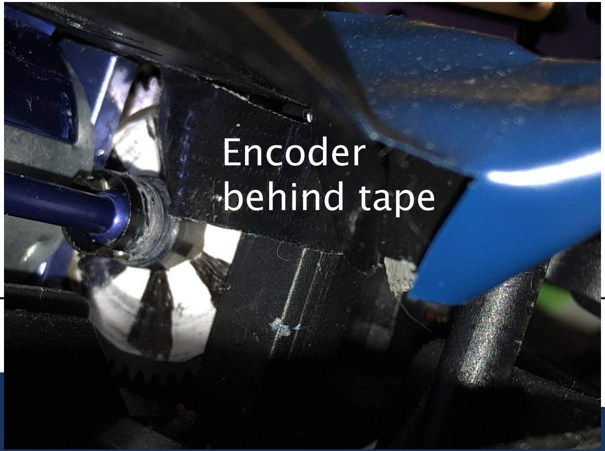
Car Photos

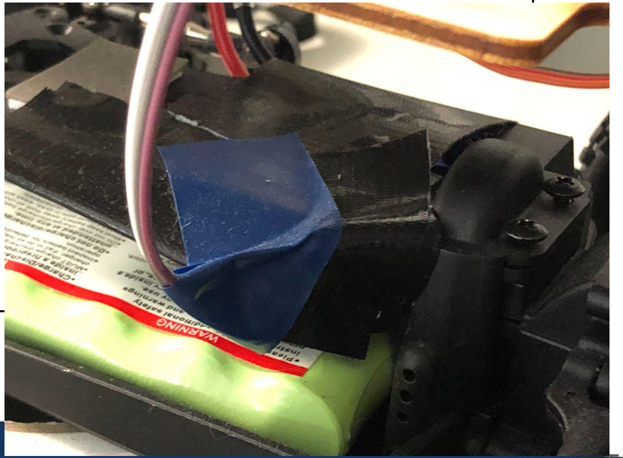




Velocity Encoder Design - Mount

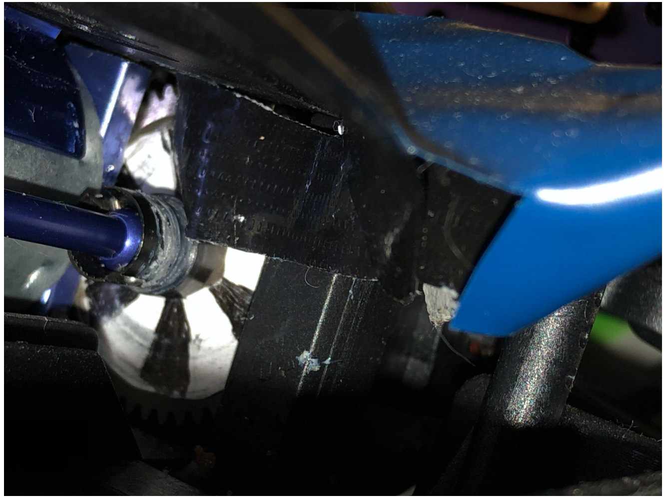
- **Initial mounting plan:**
  - Mount encoder disc to inside of wheel, mount sensor with tape
  - Would have to deal with car suspension, generally bad idea
- **Final mounting plan:**
  - Mount encoder disc to drive gear with superglue, mount sensor with tape
  - Much easier to implement/more stable
  - Tape blocks out excess light (common issue)

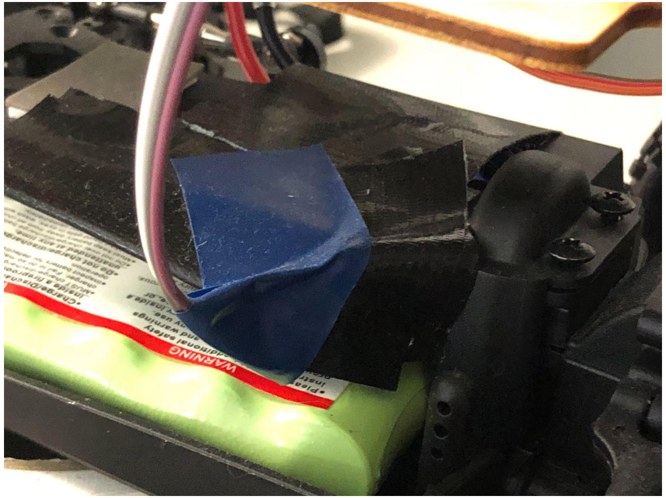




Berkeley
UNIVERSITY OF CALIFORNIA

Velocity Encoder Design - Shielding

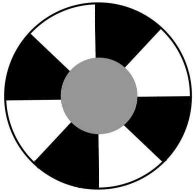




Excess light == skipped counts
Solution: shielding!

Velocity Encoder Design - Disc

- **Encoder discs:**
  - Hand cut and shaded
  - Superglued to main drive gear
- **Tested variety of encoder disc designs:**
  - 8-disc
    - Too few counts
  - **12-disc**
    - Perfect compromise
  - 16-disc
    - Sections too small



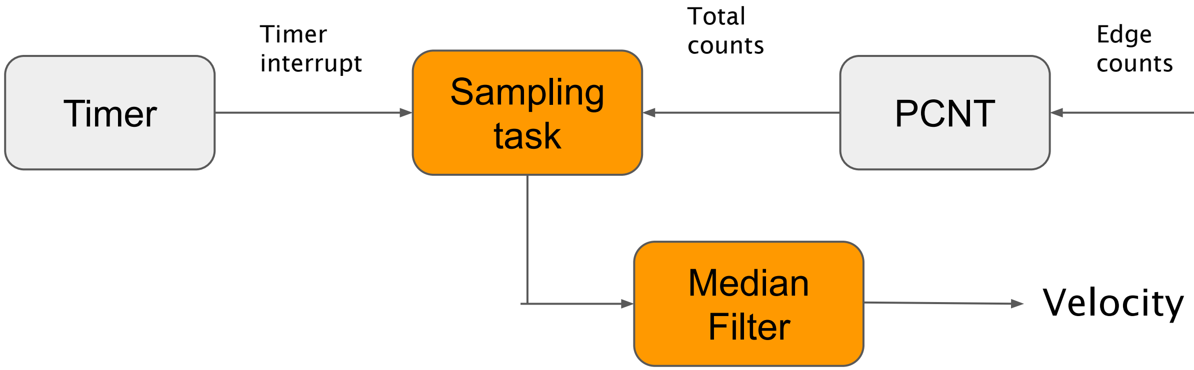
Left to right: 8-disc, 12-disc, 16-disc

Velocity Sensor - Results

- **Hardware was good!**
  - Most reliable hardware part of project
- **Software was lacking**
  - 10-20 ms too short of a period
    - Only ~25-50 cm/s velocity resolution
    - Time/frequency tradeoff (thanks Nyquist)
  - Future work: use GPIO interrupts + GPTIMER peripheral timing based approach, to measure time between ticks

Velocity Sensor - Reading

- Used **sampling approach**
  - Every 20 ms, check pulse counter value
  - # of cycles / sample converted to cm/s
- Timer subsystem used for sampling interrupts
- Pulse counter for encoder counts
- Velocity passed through 3 point median filter



Line sensor - Mounting

- Used provided mount at highest point and angle available
- Mount is unfortunately quite flimsy
    - Needs constant checking!

Berkeley
UNIVERSITY OF CALIFORNIA

Other misc onboard hardware

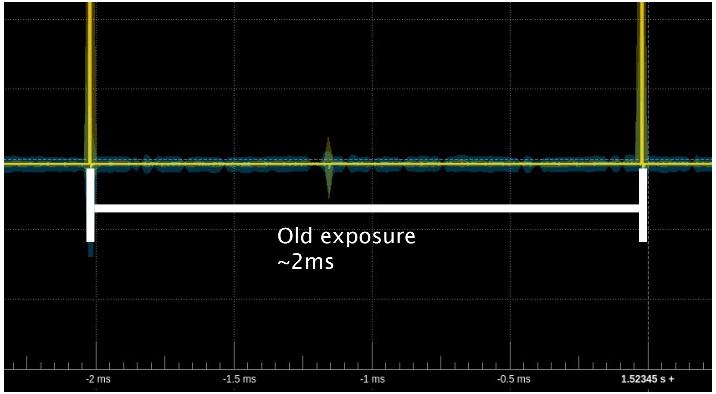
- Timers:
  - Derivative error calculation
  - Telemetry logging
  - Busywait loops in line sensor reading
- MCPWM:
  - Drives the ESC and servo inputs
  - Servo is powered from ESC
- Wifi peripherals:
  - Communications between driver station and robot

Line sensor - Reading Data

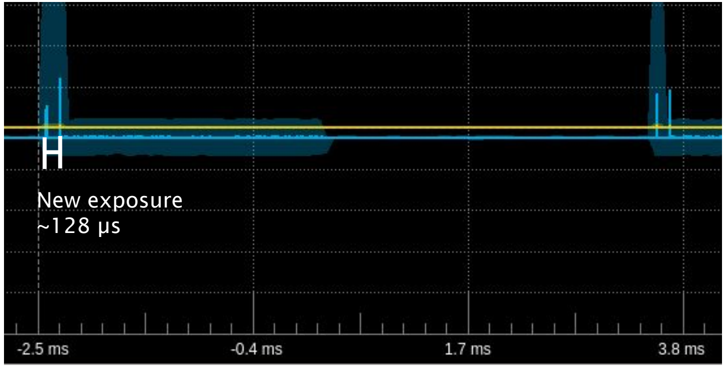
- Control signals (SI, CLK) bit-banged through GPIO
    - (TSL1401CL has ADC “SPI” protocol with exposure dependent on clock frequency)
- Data read through ADC
- **Initial strategy:**
    - Fire SI, clock and read 128 times, repeat
    - Fixed exposure time of ~2ms
    - Doesn’t work outdoors
- **Final strategy:**
    - Fire SI twice, read the 2nd time
        - Fast clocking for exposure + discard garbage, then read out data
    - Can do exposure times of <256 ns
    - Works outdoors*

(*only in shade, not direct sunlight)

voltage



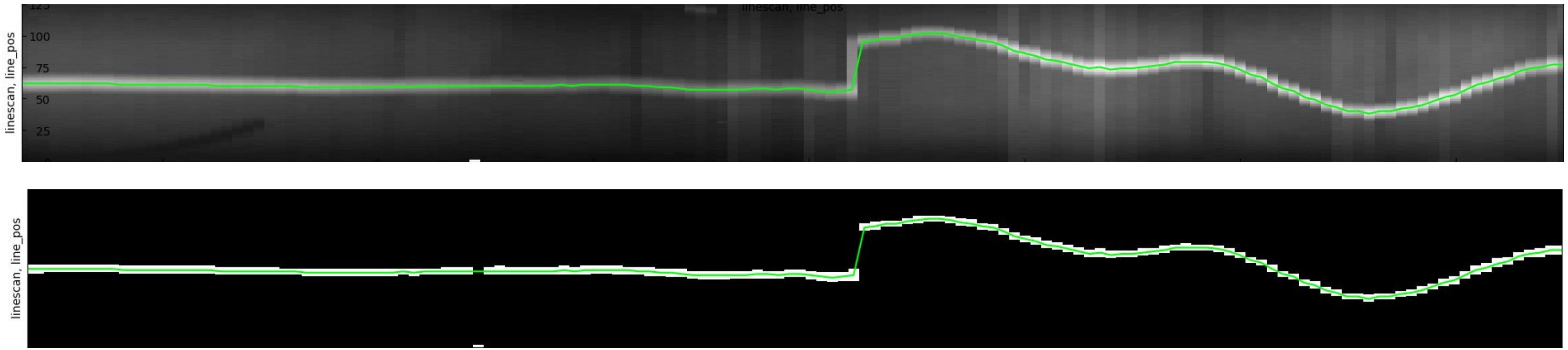
voltage



Berkeley
UNIVERSITY OF CALIFORNIA

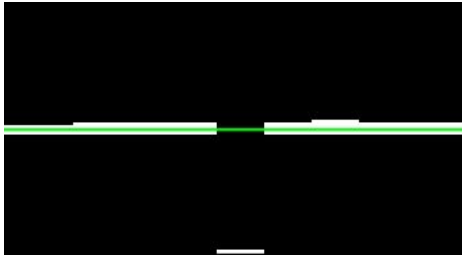
Line detection - Thresholding

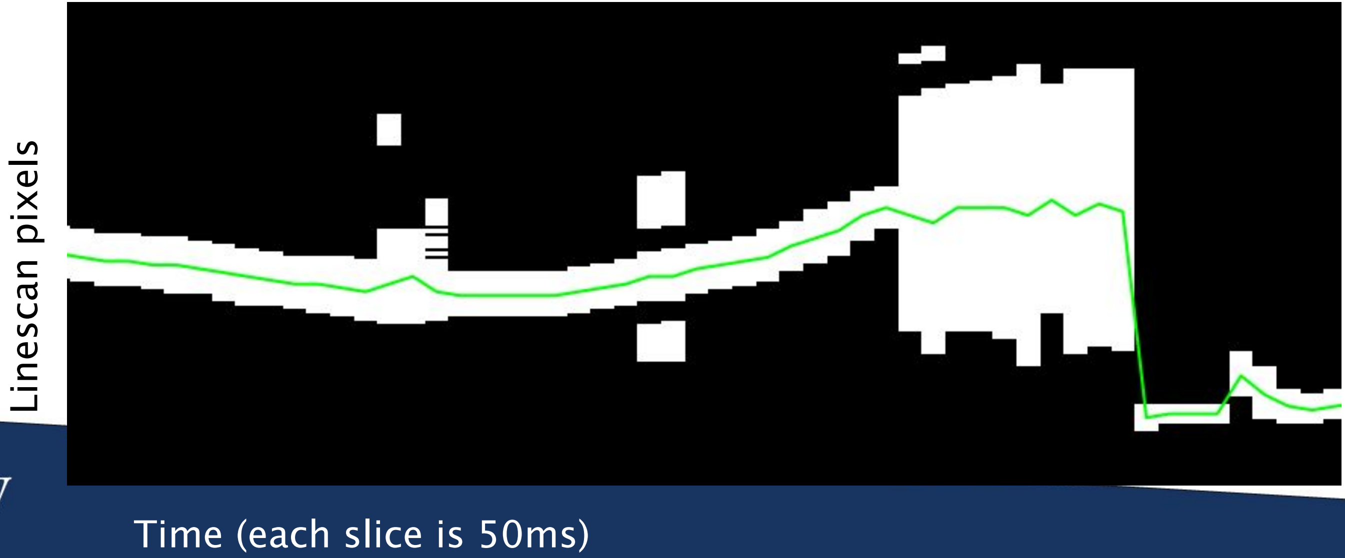
- **85% of maximum detected value AND greater than fixed min cutoff**
    - Cutoff is usually zero in practice
    - Simple yet effective



Line detection - Crossing Rejection

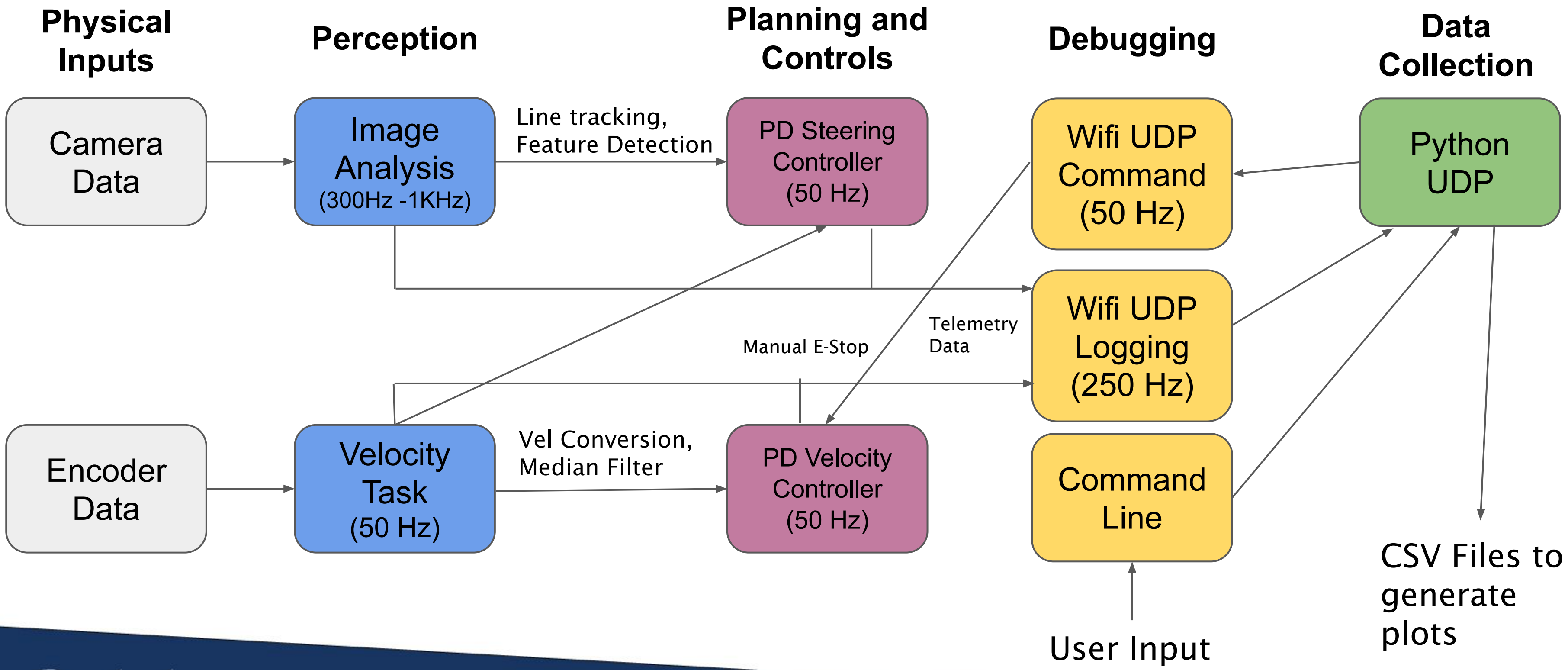
- Line algorithm organizes thresholded segments into “blobs”
  - Blob closest to previous line is likely the line
  - Center of blob is the line
  - Blob position used for stop detection
  - If no blobs, then guess the last position
  - Blobs have a minimum width





Berkeley
UNIVERSITY OF CALIFORNIA

Block Diagram for Software



Berkeley
UNIVERSITY OF CALIFORNIA

Software features

- Extensive runtime configuration system to avoid recompilation
  - Enables fast testing of experimental systems such as motion profiles, step detection
- Easy camera recalibration
  - Camera exposure, adaptive thresholds all runtime configurable
  - Commands to see camera input and statistics

```
int robot_params[ROBOT_PARAM_LEN] = {
    0,      // param[0]:       do not use
    12,     // param[1]:       velocity Kp
    25,     // param[2]:       steering Kp
    0,      // param[3]:       steering Kd
    0,      // param[4]:       (min) Threshold Light
    500,    // param[5]:       Steering Diff
    300,    // param[6]:       cross detection
    1650,   // param[7]:       velocity feedforward
    60,     // param[8]:       velocity target
    1,      // param[9]:       enable automatic velocity control if high
    8,      // param[10]:      enable force-braking the estop by setting
    1,      // param[11]:      enable stopping on natcar stop. param val
    0,      // param[12]:      enable inverting pixel colors (useful for
    85,     // param[13]:      %cutoff for argmax threshold
    128,    // param[14]:     exposure time in us
    0,      // param[15]:      (experimental) deadband railing or other
    0,      // param[16]:     (experimental) deadband error cutoff
```

Each one of these values can be modified at runtime with `paramXX VALUE`

```
Enter string "command value": cam
['cam', '0']
Enter string "command value": Reply[192.168.4.1:5555] - b'Log 8: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..\n'

Enter string "command value": camstat
['camstat', '0']
Enter string "command value": Reply[192.168.4.1:5555] - b'Log 9: min=660, max=809, median=736, thresh=687\n\n'

Enter string "command value": camexp 2048
['param14', '2048']
Enter string "command value": cam
['cam', '0']
Enter string "command value": Reply[192.168.4.1:5555] - b'Log 10: ............................................XX..............XXXXXX............XXXXXXX...........................................\n'

Enter string "command value": camstat
['camstat', '0']
Enter string "command value": Reply[192.168.4.1:5555] - b'Log 12: min=666, max=1808, median=976, thresh=1536\n\n'
```

Update rates

- Main control loop: 50 Hz (20 ms)
- Encoder sampling: every 20 ms
- Line camera: variable, depending on exposure time
  - Usually somewhere between every 1 ms-3ms (300-1000 Hz)
    - Faster than control loop to reliably detect line features at high speed
- Wifi logging (UDP send): every 4 ms (250 Hz)
  - Logging needs to be fast or telemetry will overload it!
- Wifi command receive: every 20 ms (50 Hz)

Berkeley
UNIVERSITY OF CALIFORNIA

Timing diagram

- TODO:
- (Don’t quite understand how to do the timing priority directions on this chart.)
- Also needs to be redone in Powerpoint or GIMP and with wifi + logging tasks.

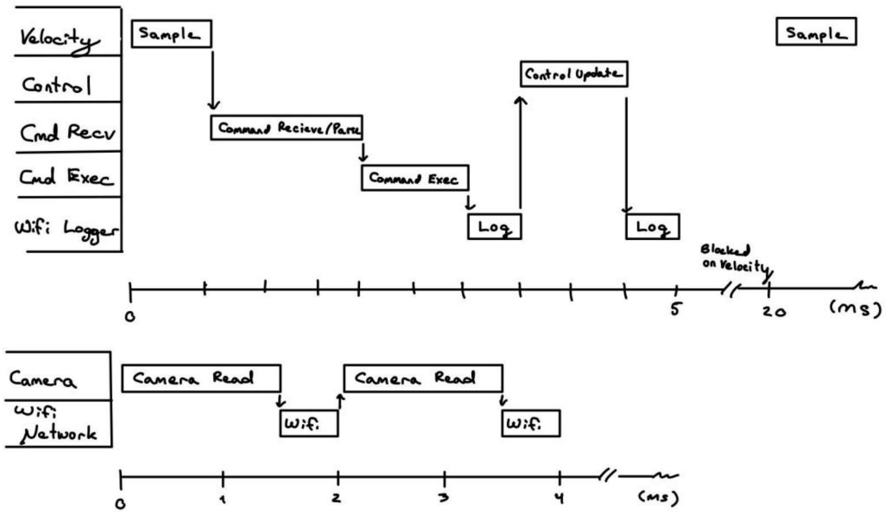


Figure 5: Software timing diagram.

Controls - Overview

- **What we used:**
  - Mostly just linear PD control
  - Experimented with motion profiles w/ step detection, didn't work well
- **Stability Problems:**
  - Oscillatory on high speed steps -- overshoot one way puts the car off track
  - kP isn't high enough on some turns
- **The compromise:**
  - Oscillations are okay as long as we still track!
    - Wiggly but still following > not wiggly but not following
    - Lose points on oscillations but not on speed
  - Jack up kP when we detect hard curves

Berkeley
UNIVERSITY OF CALIFORNIA

Controls - Implementation

- **Picking kP and kD**
  - Pick kP just high enough to track line reliably
  - Pick kD to prevent severe overshoot resulting in derailment
  - Leverage online configuration system to test tuned values

- **Error calculation**
  - Trust the line tracking subsystem
    - Responsible for stop/offcourse detection
    - And returning last known values if off course/can’t see line

Gains

- **Gains -- Velocity**
  - On real hardware, we used **Kp = 18 PWM units/(cm/s)** and no Kd.
  - Constant velocity was given to the simulation controller.
- **Gains -- Steering**
  - See the below table for details.
  - On detected hard curves, we ended up with a **Kp = 92.5 deg/m** and similar Kd to the step response.
- **Step response Gain Table:**

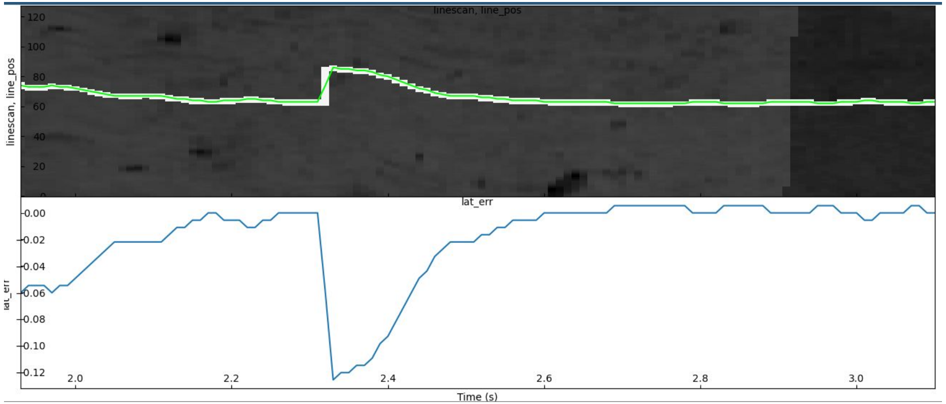
	Kp	Kd	Max Step Error	Sensed Vel	Command Vel
Real	34.7 deg/m	0.116 deg/(m*s)	12.48 cm	250 m/s	250 m/s
Simulation	400 deg/m	40 deg/(m*s)	12.5 cm	276 m/s	280 m/s

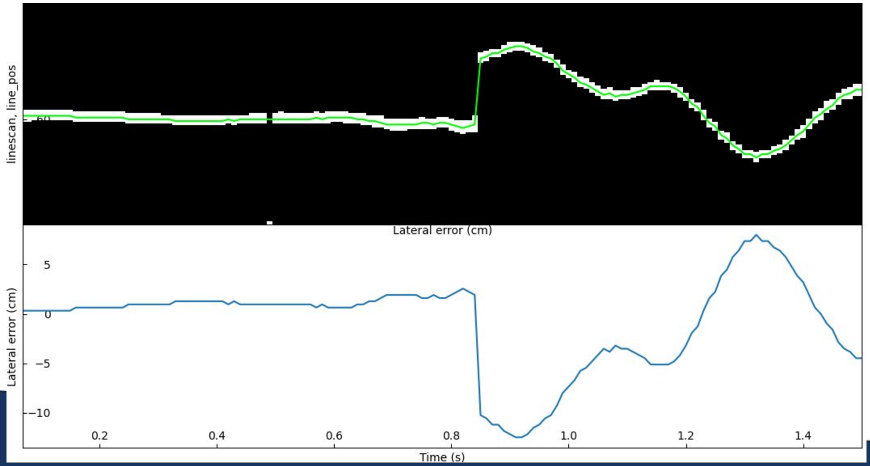
Berkeley
UNIVERSITY OF CALIFORNIA

Step Lateral Error vs. Time

- Simulation:
  X axis: time (s)
  Y axis:
  Camera scan + error (m)

- Real:
  X axis: time (s)
  Y axis: Camera scan +error (cm)





Controls - Postmortem Analysis

- **Things that worked well:**
  - Linear PD is okay especially at lower speeds
  - Controller generally strong at staying on the track

- **Things that could use improvement:**
  - Steering output nonlinearity (especially at high speed)
    - Consider arctan function like that Sp19 group?
  - Velocity control could be more even
    - Velocity readings were inaccurate with actual car speeds
  - Difficult to tell in VREP at high speeds if a simulation car’s control is “acceptable” or “oscillating”
- **Why was our car so hard to tune?**
  - The PD constants were “floats” but were casted to integers. **Oops**.

Berkeley
UNIVERSITY OF CALIFORNIA

Lessons Learnt

- **Glitches, failures, debugging issues:**
  - Figuring out the timing for SI and Clock signals without an oscilloscope (Checkpoint 4)
  - Control loop timing being too slow (Race 2)
  - Limited hours for debugging tracks outdoors (Race 2)
  - PCNT Interrupt-timed velocity control isn't a suggestion -- it's a soft requirement
- **What we wish we knew:**
  - Timing things -- FreeRTOS, priority scheduling, interrupts
  - Not initially having an oscilloscope made things really hard
  - Nonlinear PD tactics for higher speeds
- **Some advice:**
  - Wouldn't recommend this class online -- it's already hard in person
  - Test incrementally -- don't test your entire system in one go
  - Try a lot of different things -- don't fixate on one potential solution

Roles and Contributions

- **Thiti Khomin**
  - 1. Initially prototyped the SI and Clock signal timings and ADC read timings
  - 2. Initially prototyped the control loop structure
  - 3. Finely tuned Kp and Kd values in the simulator for Race 1
  - 4. Chief cardboard shading engineer for Race 2
- **Gavin Liu**
  - 1. Initially prototyped the line-detection algorithm and cross-detection algorithm
  - 2. Debugged the control loop timing and structure
  - 3. Outdoor track testing for Race 2
- **guinea wheek**
  - 1. Found a good place to mount the velocity sensor
  - 2. Debugged the initial prototype for SI and Clock signals
  - 3. Improved the line-detection and cross-detection algorithm
  - 4. Ran car during most checkpoints

Berkeley
UNIVERSITY OF CALIFORNIA

Thank you!