OpenPodCar: an open source vehicle for self-driving car research

Chris Waltham†, F. Camara*†, C. W. Fox*†
*Institute for Transport Studies, University of Leeds, UK
† School of Computer Science, University of Lincoln, UK

***Abstract*—Open source autonomous vehicle full stack software is now available (cite AUTOWARE) but there is a need for a corresponding open-source hardware platform to enable researchers to build standard setups and share research. This paper's aim is to show a process of modifying an electric pod car into a fully open-source hardware and open-source software self-driving system.**

***Index Terms*—Autonomous Vehicles, Pedestrian - Vehicle Interaction [3 maximum]**

I. Requirements

Specific requirements for such a platform are that it needs to be as low cost as possible, and easy to build. This is to enable the community to reproduce and use it. Consumer levels of safety and reliability are not required, preferring to minimise cost, through research standards of safety and reliability are required.

II. Donor vehicle

A PIHSIANG TE-889XLSN hard-canopy scooter (branded in UK as Shoprider Traverso) is used as the pod car platform. It is an electric car powered by two 12V batteries connected in series to provide 24V operating voltage and containing 75Ah. Its steering is controlled by human operated loop handle bar. Speeding and braking system are both powered by an electric motor and an electric brake via the trans-axle assembly, controlled by AC2 digital controller receiving different voltage signals to drive forward or brake. The manual speeding and braking systems were controlled by three buttons connected in series on the handle bar: A toggle switch in parallel with a resistor (10k) to choose speed mode high or low; A speed dial knob via a variable resistor (20k) to choose a maximum speed value; A throttle lever connected with a potentiometer (5k), 2.5k to 2.6k for each side to speed or brake.

III. Mechanical modification

A. Steering

To automate steering, a GLA750-P 12V DC linear actuator with position feedback is mounted under the chassis to car's front axle via bearings. This actuator has a 8mm/s full load (750N) speed and 250mm stroke length (installation length is 390mm). To access the underside of the vehicle, use two axle stands TODO photo.

There is an existing hole in the right front wheel axle. Mount the linear actuator via rear hole to the left side of front chassis and connect it through the front hole of the actuator with the hole in the car's right front wheel axle via bearings as shown in Fig. 5.

Fig. 1: Tilting the vehicle using two axle stands, to enable access to the underside. (Note also lidar mounted to roof.)

B. Sensors

1) Lidar: Velodyne16 lidar sensor. Mounted on the vehicle roof using a small optical tripod (TODO give model). Note that optical devices have unusual physical mounting standards which use Imperial rather than metrics units. This system is used by optics research based on optical table sizing. The lidar screws onto the tripod. The tripod is cabled-tied to the vehicle roof via drilled holes at locations in fig. TODO. It is mounted at a 10 degree tilt downwards (to allow pedestrians to be most clearly seen in the 16 scan lines).

2) Odometry: TODO – we need some hall effect or other shaft encoders to measure odometry.

This project has received funding from EU H2020 interACT: Designing cooperative interaction of automated vehicles with other road users in mixed traffic environments under grant agreement No 723395. Thanks to Yao Chen for making figures X and Y, and Gabriel Walton for scoping simulation tools.

Fig. 2: Underside with linear actuator added for steering.

IV. Electronic modification

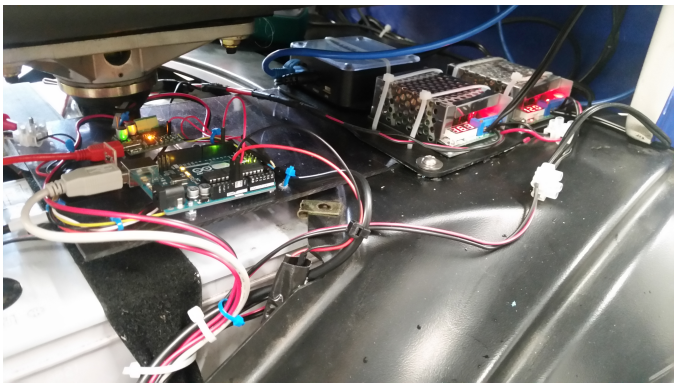

Fig. 3: Below-seat area inside cabin, with newly added boards..

A. Steering

The front wheels are steered by a Pololu (TODO give model number) PID controller, which takes serial port desired positions as input. It also takes feedback position information as an analog voltage from the linear actuator as an input. It outputs analog high-power voltages to the linear actuator.

TODO give PID settings.Windows app is needed to set them once only.

The relationship between the required central turning angle θ of the pair of front wheels and extending length l of linear actuator is,

$$\theta = \alpha - \arctan(\frac{W}{2H}) \quad (1)$$

$$\beta = \alpha - \frac{\pi}{2} \quad (2)$$

$$x = r_1 * \cos(\beta) \quad (3)$$

$$y = r_1 * \sin(\beta) \quad (4)$$

$$l = \sqrt{(x_0 - x)^2 + (y_0 - y)^2} - L + l_0 \quad (5)$$

Fig. 4: Steering console showing newly added relay (with lit LED) to dead mans handle, replacing the ignition switch (only) in the circuit.

Where r_1, x_0, y_0, W, H and L are the geometric coefficients shown in Fig.1. Among them, the value of y_0 is negative. l_0 is the initial value of the linear actuator position feedback.

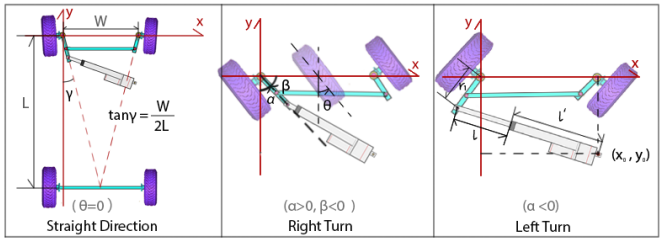

Fig. 5: The bottom view of front wheels steering relationship including geometric coefficients

Test commands can be passed to the Pololy using the commands in /tools/cmdSteer **DO NOT GIVE COMMANDS OUTSIDE RANGE 1000-25000 AS THEY HAVE DAMAGED THE VEHICLE**. TODO – WRITE SAFETY WRAPPER FOR THIS.

A non-ROS test of the C API for the Pololu is is /tools/pololuTestCSerial.

FA:cmd	Effect
2500	max right
1900	center
1000	max left

DO NOT SEND COMMANDS OUT OF THS RANGE AS THEY HAVE MECHANCIALLY DESTROYED THE SYSTEM

B. Speed

1) Dead mans handle: Deadmans handle to ignition. Replace key with relay (TODO model) and dead-mans-handle button switch (TODO model) on a 10m, 2-core cable. TODO: are any other console functions changed?

TODO edited version of wiring from shoprider manual. And photo of the real wires.

2) Speed control: TODO how to set the console switches to correct fixed settings.

Arduino UNO is used to send electric signals to vehicle's motor controller instead of using speed buttons. TODO DAC model and wiring. TODO arduino wiring.

Arduino code is supplied in the distribution (/Arduino/ThrottleControlSerial.ino). When uploaded to the Arduino (using the standard Arduino IDE running on the laptop), it provides a simple serial port API running at 112000baud (TODO other erial options.) It receives commands of the form FA:210 as speed commands. The test scripts /tools/zeroSpeed.py and /tools/testSpeed.py can be used to send example commands for debugging.

FA:cmd	Voltage	Effect
0	0	stupid fast reverse
80	0	v fast reverse (ros limit)
132	1.82	slowest reverse motion
		stop - dead zone - allows signition
201	2.71	slowest forward motion
240	2.78	v fast forward (ros limit)
255	?	stupid fast forward

To start the ignition, car safety system required the control vltage to be in the dead range. Problem is this doesnt correposnd to fixed speedbytes due to the USB power issues. But if we pick a number solidly in the center of the deadzone, such as 164, then it will be OK for most USB supplies.

when battery is flat, the voltages are lower? eg send 164 and get 1.9V instead of 2.26V ? this may result in vehicle not starting - maybe by design or accident. get beep as is outside start zone. thoery: the ardunio is getting lower power eg max 4.9V instead of 5V, which gets divided by the DAC value.

- insert Figure 2 about the wiring diagram of speed buttons and micro-controller.
- talk about safety (dead-man button).And speed code part with details.

V. ROS INFERFACE

For ROS Kinetic and Ubuntu 16.04, standard in robotics research.

TODO fig showing low level nodes

VI. SIMULATION

For Gazebo 8.6.0 (which ships with ROS Kinetic and Ubuntu 16.04, standard in robotics research).

mesh may be important for psychology research, eg size of vehicle and automation level suggest its utility functions for time and collisions as the sequential chicken model of cite(CHICKENPAPER).

TODO gazebo physics numbers

VII. ROS CONTROL AND PLANNING STACK

TODO ackerman-msgs move-base implementation

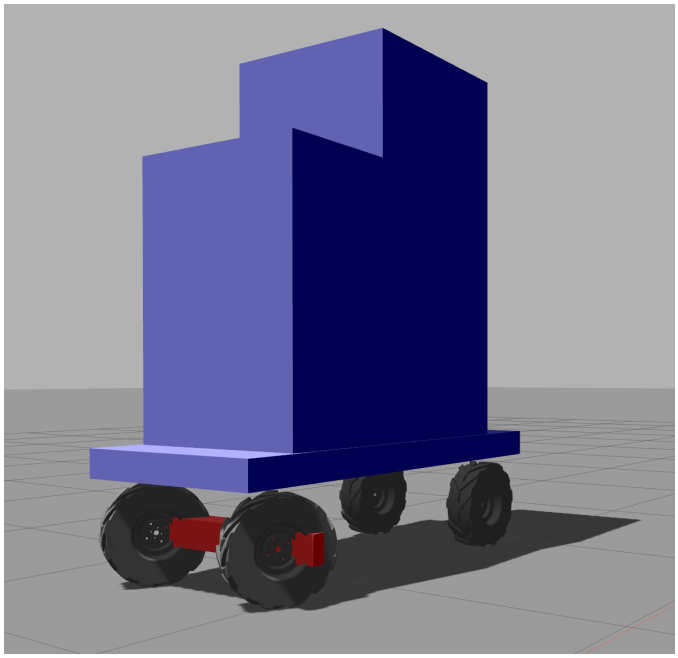

Fig. 6: Physical simulation of vehicle.

VIII. LOCALISATION AND MAPPING SYSTEM

OrbSLAM – based on Octomap, adding mono visual features. Octomap can also be used with any other point cloud data eg lidar.

NDT SLAM – used at LCAS - ILIAD.

IX. USER GUIDE

A. Power up

Check that the vehicles original lever for auto-manual is set to auto (DOWN). It is on the main motor, under the vehicle at the rear left, colored red. Requires some force to move it.

Power on the vehicle using the original on-off switch located under the seat on the left. It is marked ON-OFF.

Power on the modified electronics using the new toggle switch. (This lights LEDs on the DCDCs and Pololu, and the lidar makes a whirring sound).

Check that the batteries are charged (use a multimeter across one of the DCDC converters, need to see 24V or over. DO NOT USE THE VEHICLE IF IT IS UNDERCHARGED, THIS IS DANGEROUS.

Power on the laptop using the slider switch on its front right.

Login as user podcar, password TODO.

Type: roscd podcar

Type: roslaunch podcar podcar.launch

Use the joystick to control steering and speed.

B. Usage

X. TROUBLESHOOTING GUIDE

A. vehicle beeps continuous when press DMH and rear wheels do not move

This is due to a safety mode preventing ignition.

Fig. 7: Visual mesh simulation of vehicle.

Check: is the manual-auto switch under the rear motor on auto?

Check: are the batteries well charged (must be 24V or over.)

Check: is the control voltage is the dead zone, it should be.

XI. Related work

About 20 references or more if you can (at least one page) about similar work, SLAM, vehicle control and simulation, open source projects.... Cite this project from UC berkeley: http://www.barc-project.com/

XII. Discussion

We hope that the platform will be useful...