OpenPodCar: an Open Source Vehicle for Self-Driving Car Research

Fanta Camara[1,2], Chris Waltham[2], David Churchill[2], and Charles Fox[1,2]

[1] *Institute for Transport Studies, University of Leeds, UK*
[2] *School of Computer Science, University of Lincoln, UK*

January 2, 2022

Abstract

Open source software for autonomous vehicle ('self driving car') control is now available but there is a need for a corresponding open-source hardware platform to enable researchers to build standard setups and share research in this field. Here we are constructing such a platform. The Lincoln OpenPodCar is a physical vehicle based on a low cost off the shelf, hard canopy, mobility scooter. It is large enough to transport one person at speeds up to 15kmh, for example for use as a last-mile autonomous taxi service from the train station to the office; or to transport up to three Deliveroo-sized freight containers similarly around a city center. But it is also small enough to make serious injury to pedestrians unlikely in the event of a collision. Together with its low costs it thus forms an ideal balance between real world utility, safety, cost and research convenience. The open source platform consists of (a) hardware components: CAD designs, sample bill of materials, and build instructions to modify the off the shelf donor vehicle for electronic control; (b) Arduino, ROS and Gazebo control and simulation software files which provide standard ROS interfaces and simulation of the vehicle; and (c) higher-level ROS implementations of standard medium-level robot control, including the movebase interface which enacts command to get the vehicle from one pose to another. The vehicle uses Ackermann steering and Reeds-Sheppe curves. Its total build cost in 2022 is around 7000USD in total, including sensors and computing equipment.

Metadata Overview

Main design files: link to repository with design files and assembly instructions
Target group: researchers and hobbyists interested in autonomous vehicle research and robotics.
Skills required: Mechanical assembly – easy; electrical assembly – advanced; Software – advanced.
Replication: The work in this project is being replicated on a courrier-type manually-driven platform for a vehicle manufacturer. The current OpenPodCar is being used by some of the authors for Human-Robot Interaction experiments.

Keywords

Autonomous Vehicles, Open Source Hardware and Software.

Figure 1: OpenPodcar test drive.

Introduction

Pick a licence for the paper github: e.g. CERN hardware licence

Open source hardware

The use of open source hardware (OSH) allows for more effective and accessible sharing and collaboration among researchers [4], as well as more accessible entry-level projects such as examples from Oxer and Blummings (2009). OSH brings a number of benefits to a research community, as a common context for results and accessible platform with which to test new ideas allows for more engagement, leading to a more mature field of study.

who?

It is currently almost impossible to build any large hardware design that does not incorporate non-open designs or make use of non-open designs as tools during construction. In contrast, open source simulation platforms are widely available for autonomous vehicle research, such as CARLA [5], SUMMIT [1], FLOW[1], DEEPDRIVE[2], AirSim[17], LGSVL Simulator [15] and Gym-Duckietown [3].

The podcar is currently only "weak" OSH, like the Arduino, because it depends on the use of a patented donor vehicle and uses sensors or other components e.g. 3D Lidar that are also patented. It is intended as a basis for future work which could replace the patented components with open interfaces to progress it to "deeper" OSH.

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, though research standards of safety and reliability are required.

Donor vehicle base

A Pihsiang TE-889XLSN hard-canopy scooter (branded in UK as Shoprider Traverso, [19]) is used as the podcar platform. It is an electric mobility scooter powered by two 12V batteries connected in series to provide 24V operating voltage and containing 75Ah. In its standard configuration, the donor vehicle's steering is controlled by a human operated loop handle bar. The speed and braking systems are both powered by an electric motor and an electric brake via the trans-axle assembly, controlled by an AC2 digital controller receiving different voltage signals to drive forward or brake. The manual speeding and braking systems are controlled by three buttons connected in series on the handle bar. A toggle switch in parallel with a resistor (10kΩ) to choose speed mode high or low; A speed dial knob via a variable resistor (20kΩ) to choose a maximum

[1] https://flow-project.github.io/
[2] https://github.com/deepdrive/deepdrive

speed value; A throttle lever connected with a potentiometer (5kΩ), 2.5kΩ to 2.6kΩ for each side to speed or brake.

Open source software base

ROS, is 'an open source operating system' for robots based on a publish-subscribe pattern [13], which is the robotics community's standard interface. Building on top of ROS, the 'ROS ecosystem' is a loosely-defined collection of open-source ROS packages with variously accepted standard message formats which act as conventional interfaces to enable many community packages to work together and to act as swappable alternatives for one another.

ROS drivers are packages in the ecosystem which wrap robot hardware with standard ROS interfaces.

ROS launch files are stored configurations of ROS nodes that can be run using one command from a terminal. These are used widely by ROS packages to quickly launch the nodes required for a certain task at the same time.

tf is a ROS ecosystem library for 3D transforms. Transformas are a substantial part of ROS that allows for sensor input and other systems that require spatial frames to be contextualised by the sensors relative position within the robot. These are published on the 'tf' topic, as explained by Quigley et al (2009), and create "a dynamic transformation tree which relates all frames of reference in the system" .

Gazebo [10] is a robotics 3D physical simulation and 3D graphics display tool, separate from, but designed to integrate with ROS, and which is the robotics community standard simulator. An example of one such project that makes use of Gazebo in conjunction with ROS is reported by Qian et al 2014) which presents an example of a project in which the hardware and software are developed in parallel, making use of ROS (for providing the software interface) and Gazebo (for testing hardware that doesn't yet exist in the real world).

Related systems

SMART [CITE] is a design to modify an existing donor golf cart vehicle for automation research, developed by MIT and Singapore. The vehicle is a similar size and power to OpenPodCar but is not open source. Beetlebot [11] is a OSH system similar to OpenPodCar but for last-mile freight delivery, without human carrying capacity, and including ROS integration. Open Source Ecology (OSE) [CITE??] is an ambition project which ultimately aims to develop Deep OSH vehicles including a car and tractor, via a process of progressively deeper Shallow OSH designs. OSE is optimised for reliability and for users in developing countries so uses hydraulic power rather than electric as used in OPenPodCar. Several functioning OSH designs for large cars exist, including Apollo[3], PixBot[4]

[3]https://github.com/ApolloAuto/apollo
[4]https://gitlab.com/pixmoving/pixbot

and OSVehicle Tabby [5]. AutoRally[6] [8] developed researchers at Georgia Tech and the Berkeley Autonomous Race Car (BARC[7]) are small-scaled autonomous vehicles for research, but only BARC includes ROS integration.

Autoware [9] is a heavyweight open source software project to construct a ROS based automation stack for large on-road cars with power on-board computers. The state of Georgia, in the USA, provides a level 3 open-source autonomous vehicle based on a Ford-Edge at the Curiosity Lab at Peachtree Corners [8], where technologists and researchers can use the platform in the smart city environment free of charge.

OSH and OSS are inherently cooperative design processes and legal structures, so it is possible that components from all of the above and from podcar will be interchanged and evolve together in the future.

https://ieeexplore.ieee.org/document/8926794

https://ieeexplore.ieee.org/document/8813784

Overall Implementation and design

Mechanical modification

Steering

To automate steering, a Gimson GLA750-P 12V DC linear actuator with position feedback is mounted between an anchor on the underside of the chassis and the 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 as in fig. 2. There is an existing hole in the right front wheel axle. Mount the linear actuator via rear hole to the left side of the 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. 3 and fig. 13.

Sensor installation

Lidar option A Velodyne16 lidar sensor is 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).

[5] www.openmotors.co/tabbyevo/
[6] https://autorally.github.io/
[7] www.barc-project.com/
[8] https://www.peachtreecornersga.gov/businesses/curiosity-lab-at-peachtree-corners

Figure 2: Tilting the vehicle using two axle stands, to enable access to the underside. (Note also lidar mounted to roof.)

Depth camera option A lower cost alternative to lidar is to use a stereo camera for point cloud sensing. In this option, a StereoLabs ZedCam is mounted similarly on the vehicle roof.

TODO Odometry TODO – we need some Hall effect or other shaft encoders to measure odometry.

Electronics and Firmware

Initial design

New vehicle electronics are added on an acrylic board mounted under the seat of the vehicle, as shown in fig. 6 and 13 and mounted as in fig. 13. Details are described below. Where different voltage power supplies are required, DCDC converters are used and mounted in the same way.

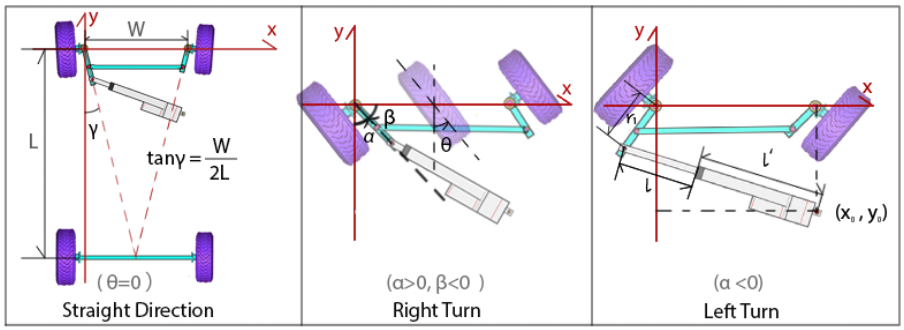

Figure 3: The bottom view of front wheels steering relationship including geometric coefficients

PCB design

To robustify the initial electronics modifications, we further designed a PCB (Printed Circuit Board). The PCB is composed of two DC-DC Buck converters with XL4016 regulator, an Arduino Uno, a MCP4725 DAC (Digital-Analog Converter), a Pololu JRK 21v3 motor controller with position feedback, 2 resistors. We 3D printed parts to support the mounting of the LCD and the 3D LIDAR.

Steering control system

The front wheels are steered by a Pololu JRK21v3 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,

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

$$\beta = \alpha - \frac{\pi}{2} \tag{2}$$

$$x = r_1 * \cos(\beta) \tag{3}$$

$$y = r_1 * \sin(\beta) \tag{4}$$

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

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.

Diagnostic test commands can be passed to the Polulo using the commands provided in /tools/cmdSteer. *Do not give commands outside the range 1000-25000 as they have damaged the vehicle.*

Figure 4: Underside with linear actuator added for steering.

TODO – WRITE SAFETY WRAPPER TO LIMIT THIS RANGE..

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

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

Do not give commands outside this range as they have mechanically destroyed the system.

Speed controller system

An Arduino UNO [12] is used to send electric signals to the vehicle's motor controller in place of the donor vehicle's paddle controller's potentiometer. An Adafruit MCP4725 DAC is connected to the Arduino as in fig. 6, and used to send clean analog speed command voltages to the donor vehicle's internal controller.

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

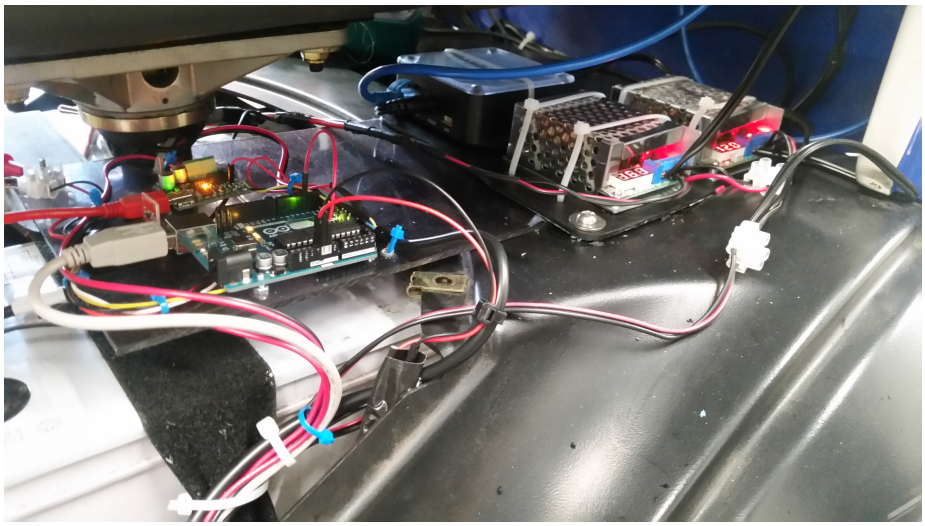

Figure 5: Below-seat area inside cabin, with newly added boards..

commands. The test scripts /tools/zeroSpeed.py and /tools/testSpeed.py can be used to send example commands for debugging.

Command	Voltage	Effect
FA:0	0	stupid fast reverse
FA:80	~ 0.9	v fast reverse (ros limit)
FA:132	~ 1.5	slowest reverse motion
		dead zone - allows ignition
FA:170	1.9	stop - zero/home position
FA:201	~ 2.3	slowest forward motion
FA:240	~ 2.7	v fast forward (ros limit)
FA:255	~ 3.0	stupid fast forward

To start the ignition, the car safety system requires the control voltage to be in the dead range. A problem is that this doesn't correspond precisely to any fixed speed bytes, due to the USB power issues. But if we pick a number solidly in the center of the dead zone, such as 164, this will work for most USB supplies. (i.e. when the vehicle's battery is flatter, the voltages provided to USB power by it are lower. For example, we might send 164 and get 1.9V instead of the usual 2.26V.) This may result in the vehicle not starting - maybe by design or accident. (This produces a beep as is outside the start zone.) The Arduino is getting lower power eg max 4.9V instead of 5V, which gets divided by the DAC value. To deal with these instabilities, we added a potential divider at the battery to check the voltage and control the podcar accordingly, as in Fig. 11. We provide a "BV" command in the Arduino serial protocol which allows callers to request the current battery voltage. This can then be used by the higher level (Python) systems to decide what speed bytes to sent, including compensating for the floating dead zone.

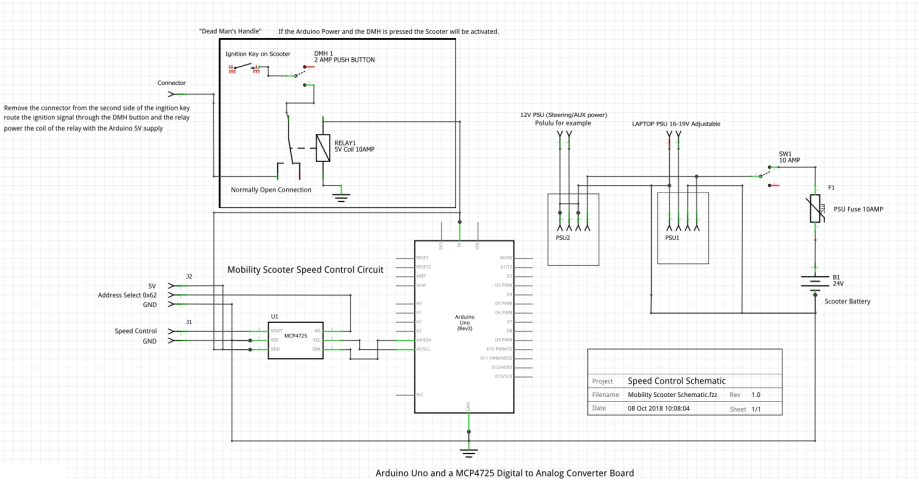

Figure 6: Circuit diagram for electronic modifications.

Physical vehicle software interface (ROS)

A ROS interface to and from the physical vehicle is provided as illustrated in fig. 15 and described below. This runs on a Panasonic CF-19 Toughbook computer mounted on-board the vehicle (and powered from a DCDC converter from the vehicle battery) running Xubuntu 16.04 (Xenial) and ROS Kinetic.

Control system interface

The system expects to hear two incoming ROS control messages: speedcmd_meterssec and wheelAngleCmd, which contain single floats representing the desired speed in meters per second, and the desired front wheel orientation in radians respectively.

These two messages are received by ROS nodes speedm2arduino and wheelAngle2Pololu, which are ROS drivers for the Arduino speed controller and the Polulo steering controller respectively.

Manual joystick control

Converters from a standard ROS USB joystick driver node to the speed and angle command interface messages are provided, by joystick2speedms and joystick2wheelAngle. These use the y axis of a joystick for speed and x for steering.

Lidar option

Velodyne works with the velodyne ROS package. But it needs to be set up so we can talk to the Velodyne over ethernet. The laptop must be on a wired

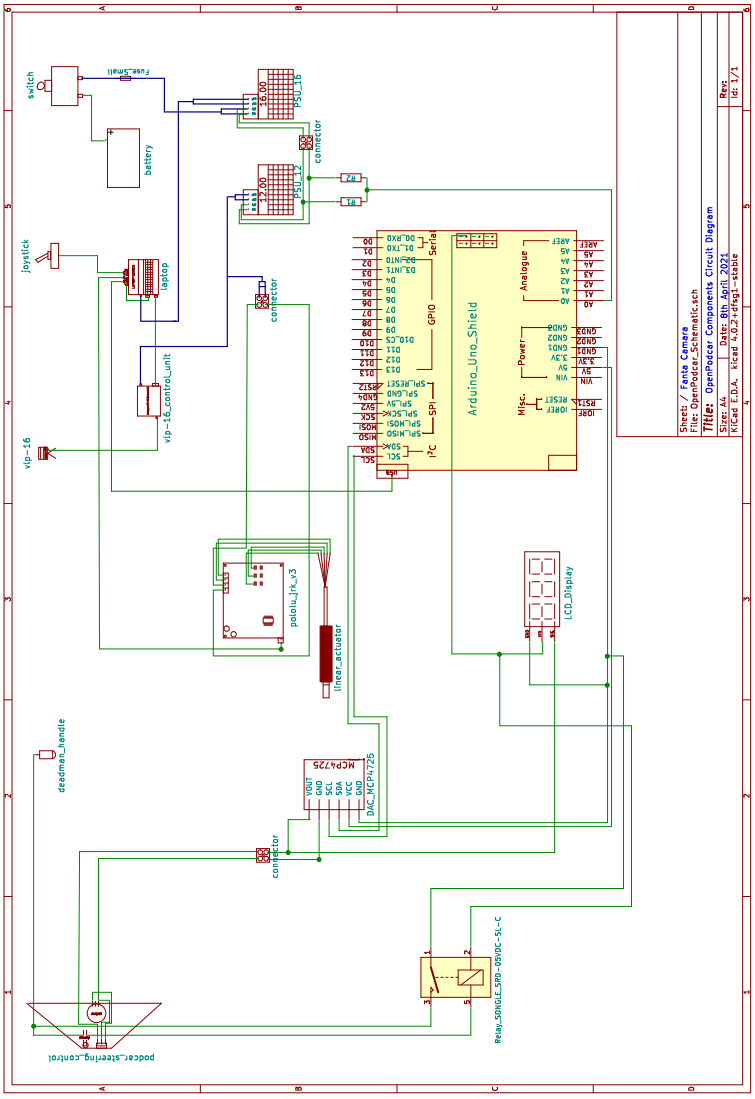

Figure 7: Complete Circuit Diagram for electronic modifications.

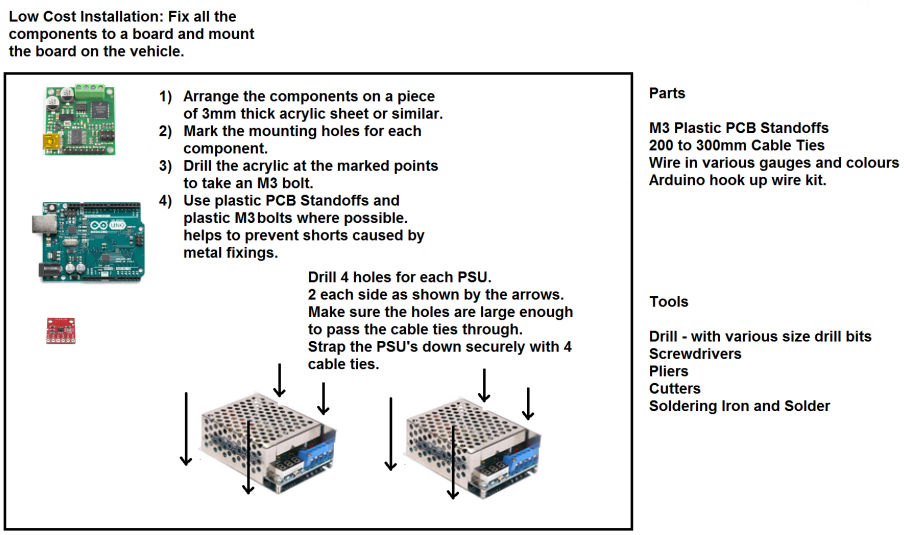

Figure 8: Drilling.

network, not wifi. The IPs must be configured as in the velodyne docs TODO details/refs. The lidar IP is 192.168.1.201.

Universality of the design

One of the biggest advantages of OpenPodCar is that the mechanical, electronic and software components can be easily ported to other vehicles/platforms and only require small changes in the software side to adapt and fix some parameters specific to the new vehicle requirements. The authors are currently replicating such a portability to another type of vehicle.

(2) Quality control

Safety

Electronics

- Attached a fuse to the batteries for protection

- Tested the PCB and each component on it before incorporating it to the vehicle

Safety System for Steering Control System

We included limitations in the steering controller for the linear actuator ranges, that will allow the vehicle to only accept and execute input values within the range that will keep the mechanical mounting safe.

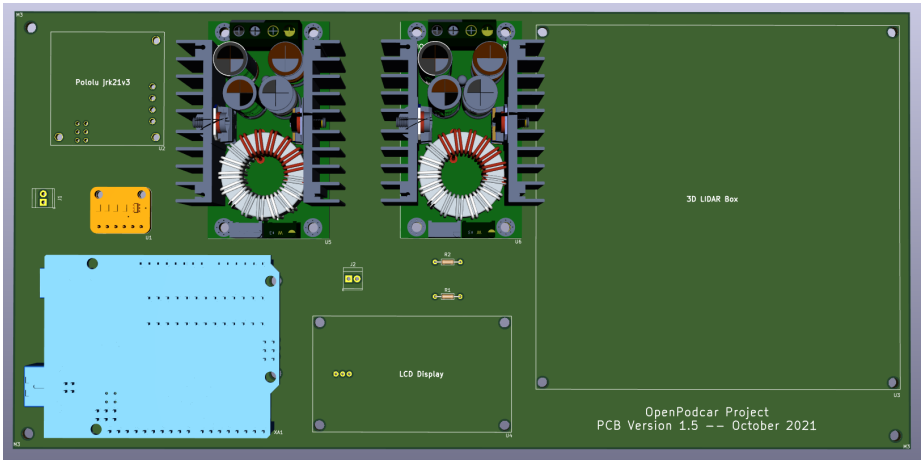

Figure 9: PCB Design.

Safety System for Speed Control System

Autonomous vehicles can present a significant hazard to people and the environment in which they operated. Damage to surroundings and possible injury to operators and bystanders could result from inappropriate use or malfunction. It is essential that a suitable emergency stop system is implemented in all autonomous vehicles. Given the development platform nature of the OpenPodCar, a safety mechanism which stops the vehicle under fault conditions is an especially important part of the design. The design thus requires a series of positive heartbeat signals to be present and timely and also a dead-mans handle (DMH) to be depressed by a human experimenter at all times, in order for a hardware relay to actively continue to supply power from the vehicle's batteries to all other systems. The relay will naturally cut out if these signals are absent for any reason, including failures in the safety systems themselves.

Implementing and testing this safety system should be undertaken with the drive wheels of the vehicle raised off of the ground, allowing for checks to be made of the DMH without the risk of the vehicle speeding off out of control.

The speed controller on this particular mobility scooter has a dead zone at around 2.5V on its speed input signal which corresponds to drive wheels stopped. Above the dead zone and up to 5V being forward control values and 0V to below the dead zone being reverse control voltages.

These values were checked using a multi-meter prior to implementing the Arduino controller and DAC. This means that if the DAC output from the Arduino where to fail and 0V for example is applied to the mobility scooters speed controller input, the vehicle would respond by spinning the drive wheels backwards at full throttle. This being a dangerous eventuality, it is very important in terms of safety to ensure this situation can not arise.

A two stage approach is used to reduce this risk. Refer to the schematic diagram DMH section in conjunction with this description.

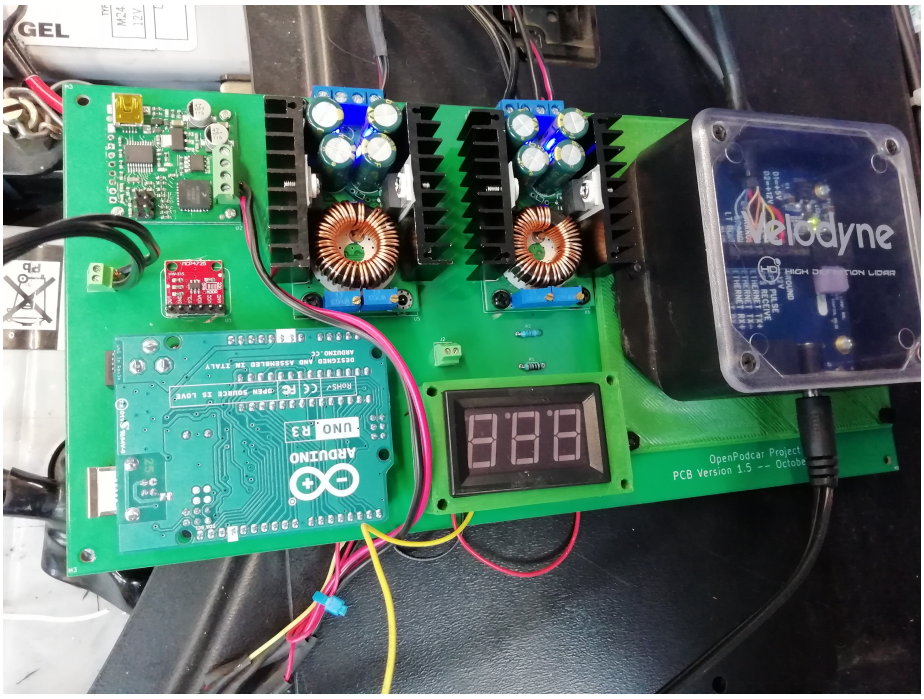

Figure 10: PCB assembled.

Stage 1 – Relay. A relay is used which interrupts the mobility scooter's key ignition circuit. If the relay is not energised by the presence of a 5V supply to the Arduino, the vehicle's movement is disabled. This effectively ensures that if the Arduino is non-functional, for example its power supply has failed or it has been unplugged from the USB port of the control PC and there is a danger that the DAC is not producing the control systems required voltage, the scooter is automatically disabled by effectively switching it off.

Stage 2 – DMH Switch. A sturdy push button is used which also interrupts the vehicle's key ignition circuit. If the PodCar operator detects any abnormality in operation during operation, he/she simply releases pressure from the DMH switch and the vehicle's movement is disabled. The DMH switch is wired in series with the relay in the key ignition circuit ensuring that if both the relay contacts and the DMH switch are closed, this is the only condition where the PodCar movement is active.

The addition of the Relay and the DMH Switch are essential for safe operation, especially where new unproven autonomous control systems are in development.

A photograph of the installed system is shown in fig. 13.

Deadman's handle to ignition. Replace key with relay (TODO model) and deadmans-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

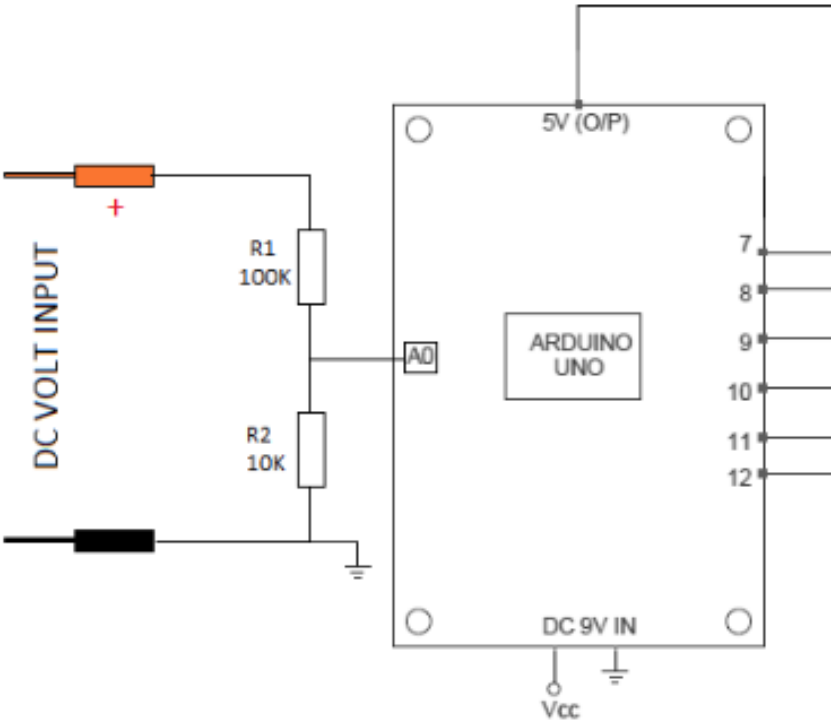

Figure 11: Potential divider linked to the battery

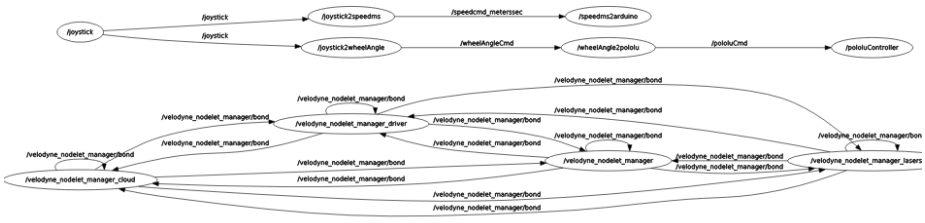

Figure 12: ROS nodes and messages for low-level control.

Figure 13: Steering console showing newly added relay (with lit LED) to dead man's handle, replacing the ignition switch (only) in the circuit.

wires.

TODO how to set the console switches to correct fixed settings.

Move_base and software safety

- collisions area

-

Calibration

If the hardware is used for measurements, please detail here how the reliability of measurements, or other hardware properties that are relevant for measurements, has been quantified and explain the results. Be clear about the processes or procedures used to compare the hardware to a standard, as well as the description of the standard calibrated against.

Detail the general procedures in place for users to calibrate their hardware before or during use. What methods can be used to relate user generated data to data from other sources?

Note: Detailed instructions belong in documentation; here, provide insight into how and why the calibration is valid.

Subsections

We encourage the use of subsections within all sections to increase clarity.

General testing

In this section, details can be provided on the testing of hardware functionalities, that are not directly essential for precision operation of the hardware in the given context (which are in turn, where applicable, handled under Calibration), such as automated movements to position the hardware, repeatability of tool exchanges, recyclability, water-tightness, weight or other possibly relevant characteristics. We encourage the authors to characterise all appropriate functionalities of the hardware, if not already described elsewhere (add reference instead). The testing should define the safe/reliable limits in which the components can be operated (e.g. step size and repeatability of linear motion, force ranges, ratio of devices with leaks when built in a workshop, etc).This will enhance the usability of the hardware or method in other contexts.

Again: Detailed instructions belong in documentation; here, provide a summary instead.

Remote Control

Before testing the full automation capability of the software, we first tested that all the components worked well when the vehicle is controlled with a joystick.

(3) Application

Use case(s)

OpenPodcar 3D Simulation

A robotics simulation of the vehicle is provided for use in Gazebo 8.6.0 under ROS Kinetic and Ubuntu 16.04 (Xenial).

The physics simulation is based on a simplified vehicle geometry with two large cuboids containing the vehicles' mass. 14 Parallel steering linkage is simulated via ODE joints forming a tracking rod. Rear drive motor and the steering linear

Change Fig. 14 with a real physical simulation of the vehicle

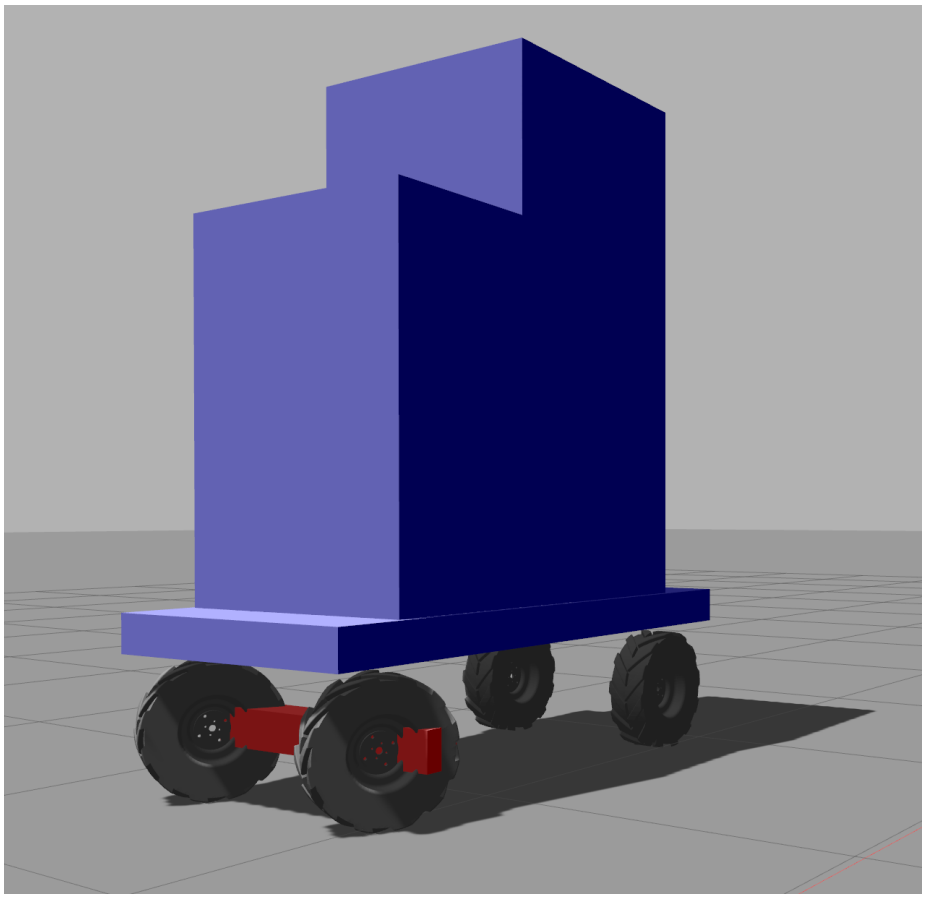

Figure 14: Physical simulation of vehicle.

Figure 15: Visual mesh simulation of vehicle.

actuator are simulated via ODE PID controllers. Wheel geometry was measured from the physical vehicle. Wheel forwards and lateral Coulomb friction coefficients were tuned by trial and error to produce realistic motion and steering. The model is represented in SDF format, which is easy portable for use in other simulation and CAD systems.

There are some key differences when it comes to actuating the tracking rod between simulation and real life. On the physical robot, a linear actuator is mounted on the underside of the vehicle and attached to the front axle, which can extend/retract to angle the wheels. However, in simulation, the pivot joints that join the front wheels and the tracking rod can have a force applied directly to them without the need for a complicated actuator. This reduces the complexity of the model without affecting its behaviour, and is still appropriately accurate to real life.

There are also some similarities, for example both steering solutions implement a PID (Proportional, Integral, Derivative) controller, that uses a proportional gain, integral function and differential function to determine exactly how much force to apply at any given time in order to keep the wheels at the desired angle. The parameters for the existing robot's PID controller have not been published, however trial and error found a proportional gain of 2, differential weight of 1 and integral weight of 0 were adequate.

The simulation implements the same ROS interface as the physical vehicle system to enable plug and plug interoperability between them. In the simulation, vehicle control messages are received, and vehicle sensor messages are sent by, a Gazebo plugin (podcarGazeboROSPlugin.cpp) which calls Gazebo's simulation functions.

Desired vehicle control commands, as for the physical vehicle, consist of (speed, angle) pairs, to command the speed of the rear wheels in m/s (speedcmd_meterssec message) and the steering angle (wheelAngleCmd message) of the front wheels in radians. The plugin and Gazebo implement these commands using simulated rotary and linear PID controllers and actuators, to rotate the rear wheels and linearly actuate the tracking rod respectively.

The plugin then publishes sensor messages corresponding to those from the physical vehicle. These include: /odometry/groundTruth, a (noiseless) standard odometry format estimate for the vehicle's pose; and /camera/image_raw and /camera_depth, the depth camera data.

A detailed graphical mesh model of the vehicle is provided for display, rather than physical simulation, purposes 15. This may be important for experiments such as virtual reality interactions between the simulated vehicle and human subjects, e.g. [2], as their psychological responses may depend on the apparent size and share of the vehicle.

A basic 3D world containing the podcar and various test objects from Gazebo libraries is provided by default as shown in Fig. 17.

Fig. 16 shows the complete ROS node configuration used during simulation, under manual joystick control.

University of Lincoln 3D Map

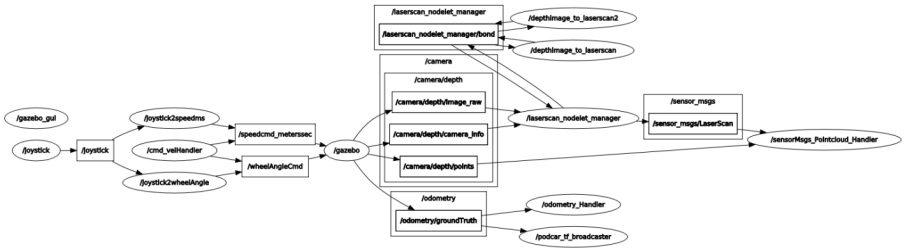

Figure 16: ROS nodes used in simulation under manual joystick control.

We used the open source Blender 3D add-on called MapsModelsImporter [CITE] to create a 3D world representative of the University of Lincoln campus, the testing area for the podcar. Before using MapsModelsImporter, RenderDoc [CITE], a free graphics debugger, is used to capture 3D data from Google Streetviews.

Path planning and control (move_base)

Path planning is the selection of an entire desired trajectory for a robot to get from a current pose to a desired pose, where a pose is a position together with a heading direction. Path planning may be split into high level and local planning, where high level makes large scale decisions such as which way to go around obstacles, while local planning refines these choices into specific curves with optimal properties. Path control (or path following) is then the real-time process of executing a path plan by interactively monitoring the robot's state, sending commands to motors, and trying to make the actualized path as close to the desired path as possible.

Movebase (wiki.ros.org/move_base) is a ROS community standard which defines a set of message formats and a framework of ROS nodes for path planning and control. This framework allows different algorithms to plug in to it. Timed Elastic Band (TEB) [16] is a local planner algorithm which has been integrated as a movebase plugin (wiki.ros.org/teb_local_planner).

Planning and control for car-like vehicles with a steering column is more complex than for differential drive (skid steer) vehicles which turn by having the wheels on one side rotate faster than the other. This is because differential drive vehicles are holonomic, meaning that in the absence of obstacles, they can move greedily from any pose to any other, while steered vehicles and non-holonomic, meaning that greedy motion is not guaranteed to reach a target pose and longer-term plans must be considered. The shortest possible path for a forward-driving steered vehicle is the Dubins path [6], which always consists of two circle arcs connected by a straight line, which is generalized to the Reeds-Shepp path when reversing is also allowed [14]. Parallel parking is a classic difficult path planning task for Reeds-Shepp vehicles.

Front-wheel-steering means that the front two wheels are used to steer the vehicle, with the rear wheels trailing and-or powering the vehicle. Ackermann steering is a special case of front-wheel-steering, in which the front wheels are

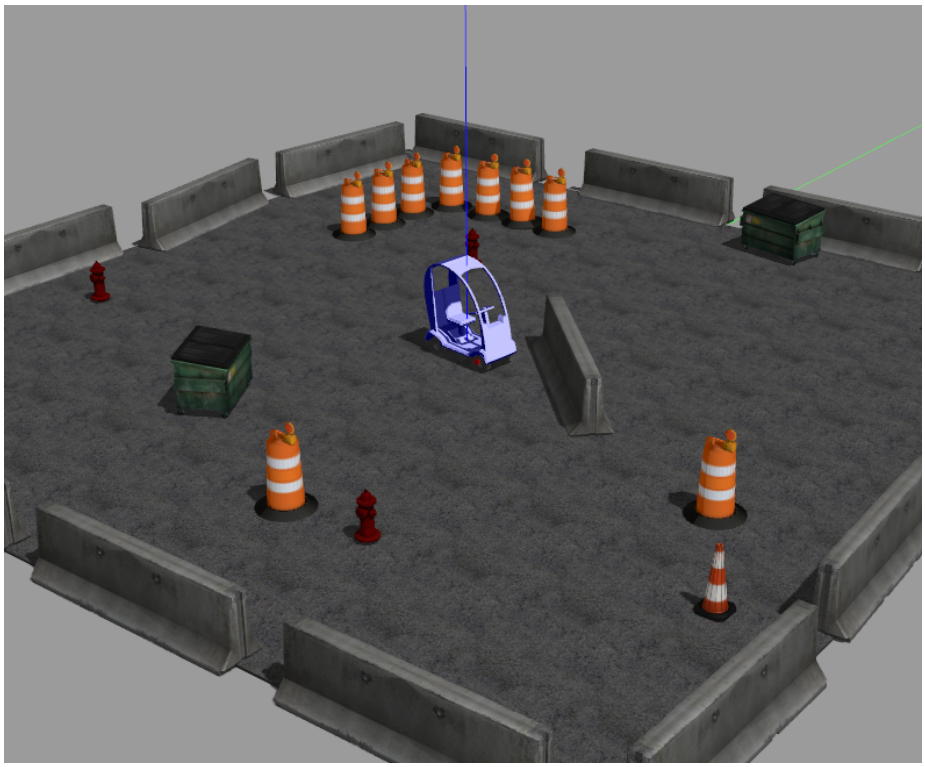

Figure 17: Initial gazebo simulation world

connected to a tracking rod via two shorter steering arms arranged so that the two steering arms form a triangle with the center point of the vehicle's rear axle when the wheels are facing forwards. This results (TODO cite proof) in the front wheels forming the correct angles (which are different from each other) needed for the vehicle to drive in circular segments, with the circle radius determined by the steering angle. Vehicles whose possible driving motions are of this form are called Dubins Cars if they can drive only forwards, or Reeds-Shepp cars if they can drive forwards and reverse. ubins and Reeds-Shepp cars have possible driving trajectories from pose A to pose B which consist of at most one circle arc segment and two straight lines to and from it. The donor podcar is a Reeds-Shepp car with Ackermann steering.

The values for parameters such as minimum turning radius were calculated from the technical specifications of the base vehicle [18].

Move Base is a set of ROS nodes that are standard for controlling robots via ROS. It enforces conventions for being able to send messages to controllers, for example the use of Twist messages. These messages are made up of 6 floats, the first 3 representing linear velocity in each axis (x, y and z; forward, left and up) with the latter 3 representing angular velocity in the same space. Move Base requires the robot to move with these given velocities when a Twist message is published to the /cmd vel topic (Marder-Eppstein, 2018). These standards ensure compatibility between packages, making the addition of existing packages

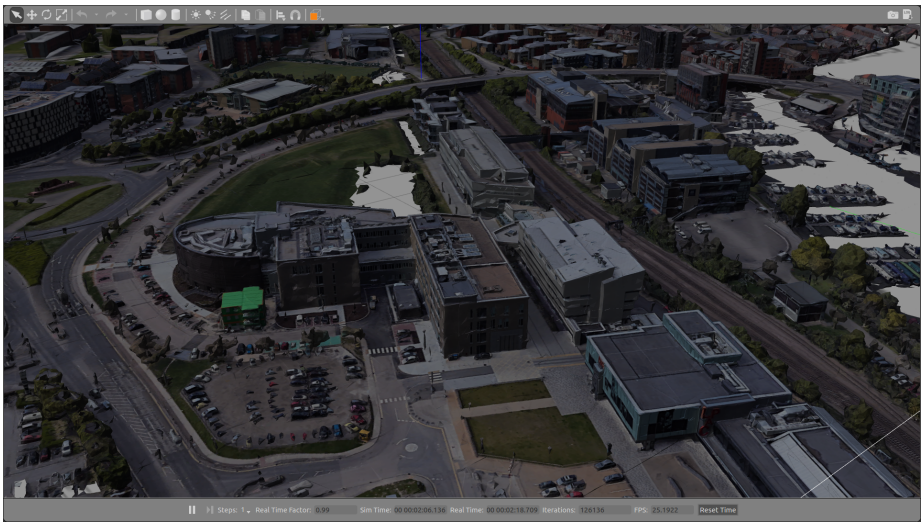

Figure 18: University of Lincoln's Gazebo simulation world

to a project much easier.

Twist messages, however, don't apply to all robots. For example, the vast majority of robots aren't able to move directly up in the world, so the linear z component of the message is ignored. A similar problem arises when considering 'car-like' steering, as they are unable to inflict an angular velocity. In this case, the angular z component is instead taken to refer to the desired angle of the steerable wheels, which resolves the incompatibility problem provided all nodes involved are aware of the difference. Alternatively, an intermediary node could be introduced that calculates the wheel angle needed to make the robot turn at a given rate, however this would be unnecessarily complex and wouldn't have the desired effect if the robot's linear velocity was 0.

Fig. 16 shows the complete ROS node configuration used during movebase, with the simulation backend. When using the physical vehicle, simulation nodes are replaced by the physical vehicle nodes as described above.

Localisation and mapping system

Simultaneous Localisation and Mapping (SLAM) [20] is the robotic task of inferring the robot's location at the same time as building a map of its environment, which is a classic 'chicken and egg' problem as the two subtasks depend on one another. Solving SLAM is an NP-hard problem but many standard approximations exist. gmapping [21] is a ROS implementation of a Rao-Blackwellized Particle Filter (RBPF)for 2D SLAM, in which "each particle carries an individual map of the environment" The information carried by each particle overlaps, and an estimation of a map can be built based on these relationships. As the robot moves around the environment, these estimations are stored, and when a 'feedback loop' is closed, the estimations cascade into a portion of the completed map. These maps take the form of 2-D occupancy grids, and can be used later

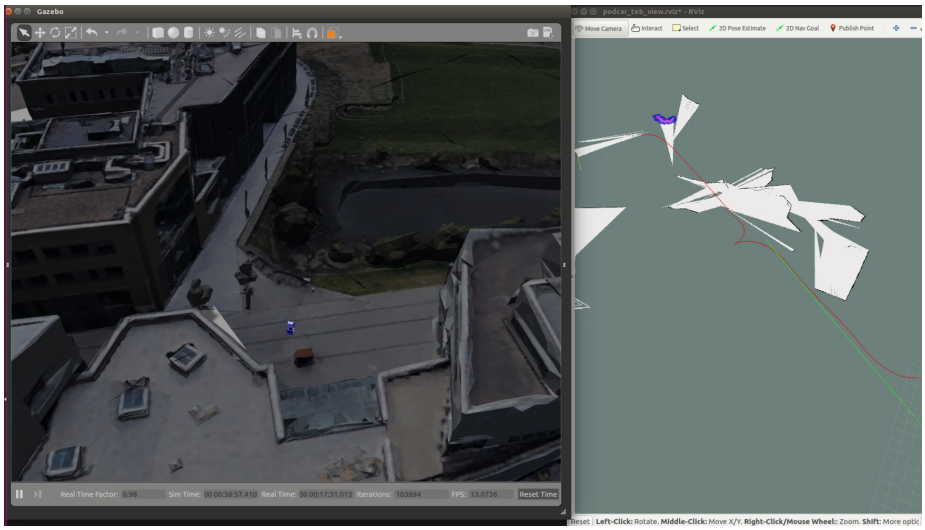

Figure 19: Podar in the University of Lincoln's Gazebo simulation world with Rviz TEB output

by the navigation stack to plan paths around the environment.

Performing SLAM in 3D is more computationally challenging, but has recently become possible due to hardware acceleration provided by parallel GPU computers. OrbSLAM is a ROS 3D SLAM based on the Octomap algorithm, which can operate with mono visual features, or lidar or stereo camera point clouds. It maintains a voxel based map of the environment (similar to maps in the game Minecraft) and is a 3D analog of gmapping.

NDT SLAM (used in Autoware and ILIAD) is another 3D SLAM implementation and algorithm based on Normal Distribution Transforms [7].

gmapping – current

TODO

Reuse potential and adaptability

(4) Build Details

Availability of materials and methods

Ease of build

Operating software and peripherals

The work mainly requires Arduino IDE, Pololu Configuration Utility Manager, Ubuntu 16.04, Robot Operating System (ROS) Kinetic, Gazebo, KiCad (PCB

Design).

Dependencies

E.g. other hardware or software projects, modular components, libraries, frameworks, incl. minimum version compatibility.

Hardware documentation and files location:

Archive for hardware documentation and build files (required. We recommend the use the DocuBricks repository, please see author guide for criteria and alternative repositories.) Note: We require the inclusion of modifiable design files as well as a detailed documentation of the functionality of the hardware with assembly instructions. This will be assessed as part of the journal peer review process.

Name: The name of the archive

Persistent identifier: e.g. DOI, etc.

Licence: Open hardware license under which the documentation and files are licensed - see author guide for more information

Publisher: Name of the person who deposited the documentation

Date published: dd/mm/yy

Modifiable design files (if different from above)

Name:The name of the emulation environment

Persistent identifier: e.g. DOI, handle, PURL, etc.

Licence: Open license under which the software is licensed here

Publisher: Name of the person who deposited the documentation

Date published: dd/mm/yy

Software code repository (e.g. SourceForge, GitHub etc.) (required)

Name: The name of the code repository

Identifier: The identifier (or URI) used by the repository

Licence: Open license under which the software is licensed

Date published: dd/mm/yy

(5) Discussion

Conclusions

Conclusions, learned lessons from design iterations, learned lessons from use cases, summary of results.

Future Work

- Depth camera option

- Motor controller with OSMC

- Human-Robot Interactions

Paper author contributions

Task (e.g. design, assembly, use cases contribution, documentation, paper writing), contribution, author name.

Acknowledgements

The authors would like to thank Jacob Lord for creating the vehicle graphical mesh model, Yao Chen for scoping Dubins path methods and the mechanical design, Gabriel Walton for scoping simulation tools, Yicheng Zhang for helping with the 3D printer and many other useful tools for the PCB board.

Funding statement

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.

Competing interests

The authors declare that they have no competing interests.

References

[1] Panpan Cai, Yiyuan Lee, Yuanfu Luo, and David Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4023–4029. IEEE, 2020.

[2] Fanta Camara, Patrick Dickinson, and Charles Fox. Evaluating pedestrian interaction preferences with a game theoretic autonomous vehicle in virtual reality. *Transportation Research Part F: Traffic Psychology and Behaviour*, 78:410–423, 2021.

[3] Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, and Liam Paull. Duckietown environments for openai gym, 2018.

[4] Fisher Daniel K and Gould Peter J. Open-source hardware is a low-cost alternative for scientific instrumentation and research. *Modern instrumentation*, 2012, 2012.

[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[6] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[7] Erik Einhorn and Horst-Michael Gross. Generic ndt mapping in dynamic environments and its application for lifelong slam. *Robotics and Autonomous Systems*, 69:28–39, 2015.

[8] Brian Goldfain, Paul Drews, Changxi You, Matthew Barulic, Orlin Velev, Panagiotis Tsiotras, and James M Rehg. Autorally: An open platform for aggressive autonomous driving. *IEEE Control Systems Magazine*, 39(1):26–55, 2019.

[9] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

[10] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[11] Anh Nguyen, Erman Tjiputra, and Quang D. Tran. Beetlebot: A multipurpose ai-driven mobile robot for realistic environments. In *Proceedings of the Third conference on UK Robotics and Automation (UKRAS)*. UKRAS, 2020.

[12] Jonathan Oxer and Hugh Blemings. *Practical Arduino: cool projects for open source hardware*. Apress, 2011.

[13] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[14] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[15] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. *arXiv preprint arXiv:2005.03778*, 2020.

[16] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. In *2013 European Conference on Mobile Robots*, pages 138–143. IEEE, 2013.

[17] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[18] Shoprider. Flagship luxury scooter model: Te-889xlsnuser manual. Technical report.

[19] Shoprider. Shoprider flagship luxury scooter model: Te-889xlsn user manual. *Manual*, 2016.

[20] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[21] Barry Loh Tze Yuen, Khairul Salleh Mohamed Sahari, and Zubaidi Faiesal Mohamad Rafaai. Improved map generation by addition of gaussian noise for indoor slam using ros. *Journal of Robotics, Networking and Artificial Life*, 4(2):118–123, 2017.

Copyright notice

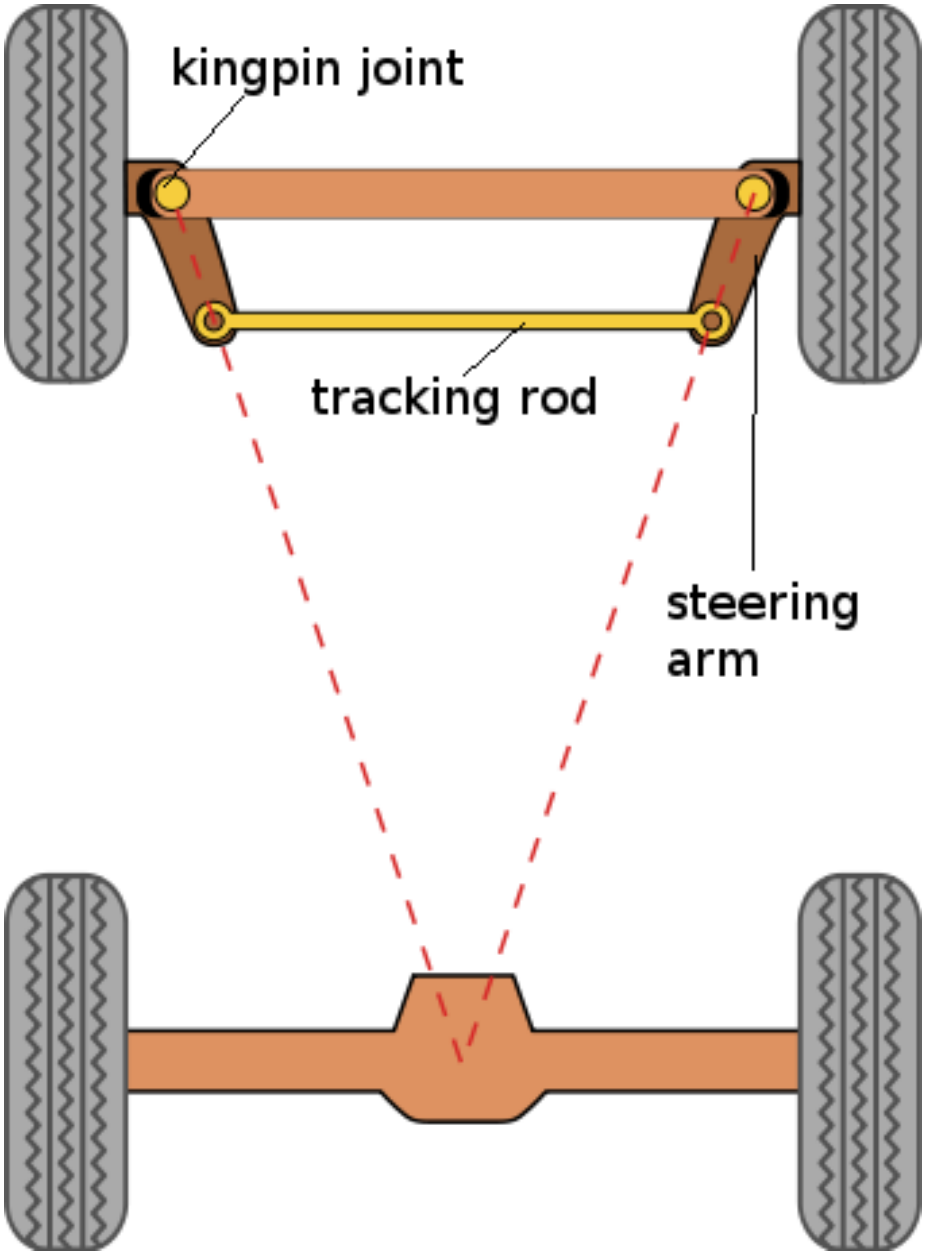

Figure 20: Ackermann steering (Source: Wikipedia; Ackermann Steering; Creative Commons.

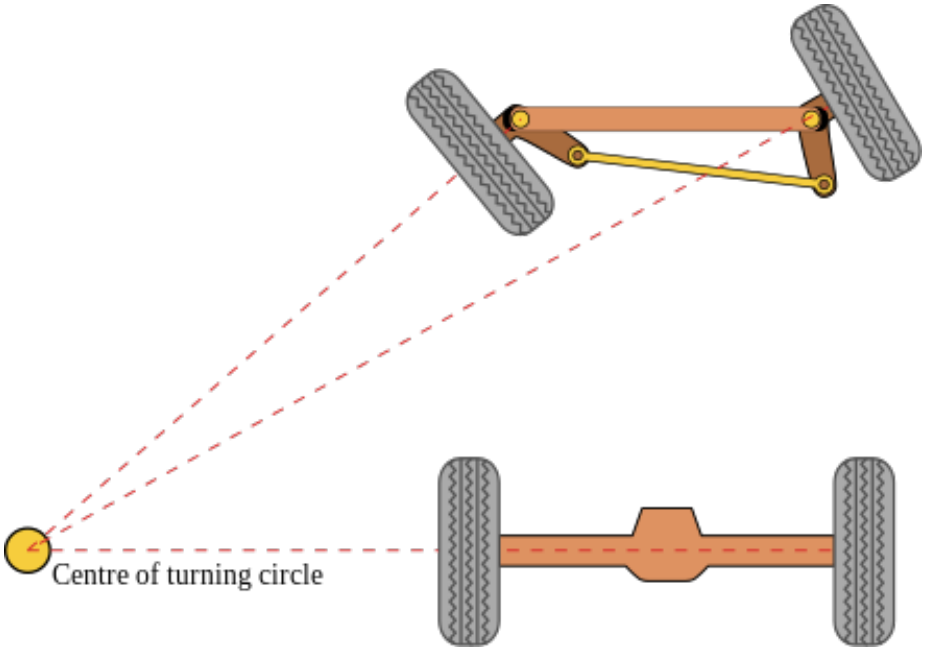

Figure 21: Ackermann steering (Source: Wikipedia; Ackermann Steering; Creative Commons.

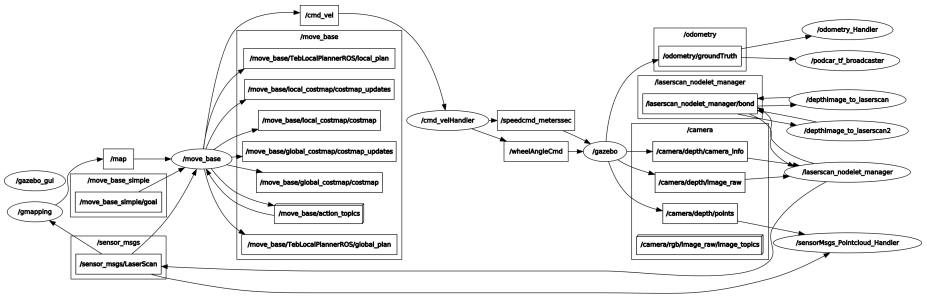

Figure 22: ROS nodes used in simulation under movebase control.