Hindawi
Journal of Robotics
Volume 2018, Article ID 4907536, 10 pages
https://doi.org/10.1155/2018/4907536

Research Article

A Low-Cost Model Vehicle Testbed with Accurate Positioning for Autonomous Driving

Benjamin Vedder,[1] **Jonny Vinter**,[1] **and Magnus Jonsson**[2]

[1]*Department of Electronics, RISE Research Institutes of Sweden, Sweden*
[2]*School of Information Technology, Halmstad University, Sweden*

Correspondence should be addressed to Benjamin Vedder; benjamin.vedder@ri.se

Received 13 June 2018; Accepted 18 October 2018; Published 18 November 2018

Academic Editor: Huosheng Hu

Accurate positioning is a requirement for many applications, including safety-critical autonomous vehicles. To reduce cost and at the same time improving accuracy for positioning of autonomous vehicles, new methods, tools, and research platforms are needed. We have created a low-cost testbed consisting of electronics and software that can be fitted on model vehicles allowing them to follow trajectories autonomously with a position accuracy of around 3 cm outdoors. The position of the vehicles is derived from sensor fusion between Real-Time Kinematic Satellite Navigation (RTK-SN), odometry, and inertial measurement and performs well within a 10 km radius from a base station. Trajectories to be followed can be edited with a custom GUI, where also several model vehicles can be controlled and visualized in real time. All software and Printed Circuit Boards (PCBs) for our testbed are available as open source to make customization and development possible. Our testbed can be used for research within autonomous driving, for carrying test equipment, and other applications where low cost and accurate positioning and navigation are required.

1. Introduction

It is common to use model cars for automotive research, and several studies are published in that field. For example, research within vehicle platooning has been carried out on model cars equipped with floor marking and distance sensors [1, 2], with the goal of developing Model Predictive Control (MPC) algorithms for controlling the distance between adjacent vehicles. Model cars have also been used to develop obstacle avoidance algorithms for mobile robots [3] and in student projects to teach them about autonomous driving [4]. These projects are specific and aimed at certain tasks, and the hardware and software are not available to replicate for use within other areas.

The need for a generic model vehicle platform for education and research within autonomous driving is recognized in the community, and several attempts at answering that need have been made. A project named MOPED [5] provides a model car that has three Raspberry Pi single board computers [6] connected over ethernet to simulate part of the complexity of a modern full scale car. Two of the computers on the MOPED model car run AUTOSAR [7] while the third one runs the default Raspbian Linux distribution (https://www.raspberrypi.org/downloads/raspbian/) that comes with the Raspberry Pi. The reason for using AUTOSAR is to represent a software stack similar to the one on a full scale car; however it requires software tools that are not available as open source or freeware for developing the AUTOSAR portions of the software. Further, the MOPED model car only provides low level control functions for the motor and steering servo, meaning that the users have to implement trajectory following and positioning algorithms themselves, as well as equipping the car with the necessary sensors. Gulliver [8] is another initiative that addresses the need of a miniature vehicle platform in research and education environments. In their publication, the authors describe high level algorithms for handling different traffic scenarios that can be used given access to a model vehicle that can follow trajectories. While these projects are an aid in education and research about autonomous driving, a significant amount of work is still required from the researchers or students to get a self-driving model car up and running. Note that, with self-driving in this context, we refer to the ability to follow a predefined trajectory accurately and repeatedly, which has been one of the goals in our work as explained below.

Our work focuses on providing a hands-on hardware and software testbed that can be used to build self-driving model vehicles with minimal effort. The goal is to provide the possibility to get a model vehicle up and running and follow a custom trajectory autonomously with 5 cm or better accuracy in just one day of work, given that the user has a background within electronics. To achieve this, we have developed low-cost hardware and both embedded and desktop software controlling model vehicles with Ackerman steering that can be modeled with a bicycle model [9]. Our testbed also has support for Hardware-in-the-Loop (HIL) testing [10, 11] by simulating parts of the vehicle dynamics, which is useful during development and automatic testing. Thus, the contribution of this work is to answer to what extent the aforementioned goal can be achieved with low-cost hardware, provide help for other researchers who want to implement their own self-driving model car, and answer what performance and accuracy can be expected. All software and hardware design for our testbed are available on github (https://github.com/vedderb/rise_sdvp) for making it possible to study and extend our platform. To our knowledge there is nothing available today that can fulfill the goal that we have for our tested, and as expressed in the aforementioned studies [5, 8] there is a need for that in the education and research communities.

In addition to usage as a research and education platform, our testbed can be used in measurement and data collection applications. For example, we have used it to pull a trailer with different radar units to be characterized around a variety of radar targets. This way the radars can be moved along a predefined trajectory around the targets, while the measurements they take are logged together with accurate position stamps. Another possible application that we have been considering is using model cars based on our testbed to carry light sensors and map the light intensity of artificial lighting in outdoor environments. The open source nature of the software in our testbed and the versatile visualization tools that are part of it make this a relatively simple task.

The remainder of this paper is organized as follows: in Section 2 we describe the architecture of our testbed and in Section 3 we describe how positioning is performed. Section 4 describes our trajectory following approach and in Section 5 we present our conclusions from this work.

2. Architecture Overview

Our testbed consists of a control Printed Circuit Board (PCB) we have developed that can be connected to a VESC (https://vesc-project.com) open-source motor controller over Controller Area Network- (CAN-) bus. Together with a battery and a Global Navigation Satellite System (GNSS) antenna they can be connected to the Permanent Magnet Synchronous Motor (PMSM) and steering servo suitable for a model car. The control PCB together with a

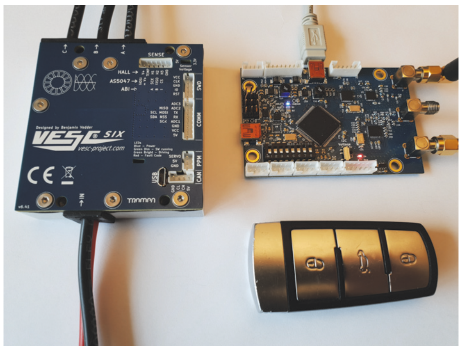


Figure 1: Our custom control PCB and our VESC 5 kW motor controller next to a car key for size comparison.

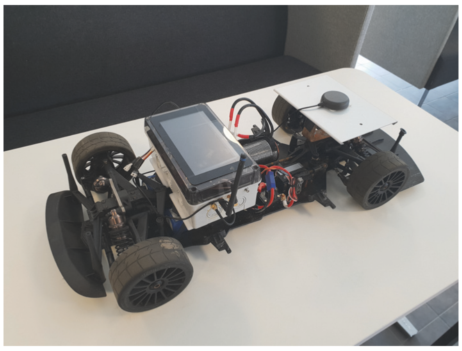


Figure 2: The 1:6 scale model car in our testbed.

5 kW VESC motor controller is shown in Figure 1 together with a car key for size comparison. Another unit of the same control PCB configured as a communication interface can be connected to a laptop computer running RControlStation, which is the monitoring and control Graphical User Interface (GUI) for our testbed. The stationary control PCB can also be connected to a GNSS antenna and act as a Real-Time Kinematic Satellite Navigation (RTK-SN) base station for the testbed, eliminating the need for an external base station. This gives a minimal stand-alone configuration of our testbed, which is able to follow trajectories autonomously. The cost of this configuration, excluding the laptop cost, is in the range of €900 to €2000 depending on the choice of model car, battery size, VESC version, and GNSS antennas. The schematics and hardware layout of our control PCB, the control PCB firmware and the RControlStation software, and the VESC firmware and configuration software are all available on github (https://github.com/vedderb).

While the aforementioned minimal configuration is sufficient for getting everything running and carrying out experiments, a Raspberry Pi (https://www.raspberrypi.org/) single board computer can be added for remote debugging, for video streaming, and for providing WiFi or 4G cellular connectivity. Our github repository also contains a command line utility for the Raspberry Pi that among other things provides a TCP/UDP to USB bridge for communication with RControlStation over WiFi or 4G. Figure 2 shows a photo of one of our model cars and Figure 3 shows a block diagram

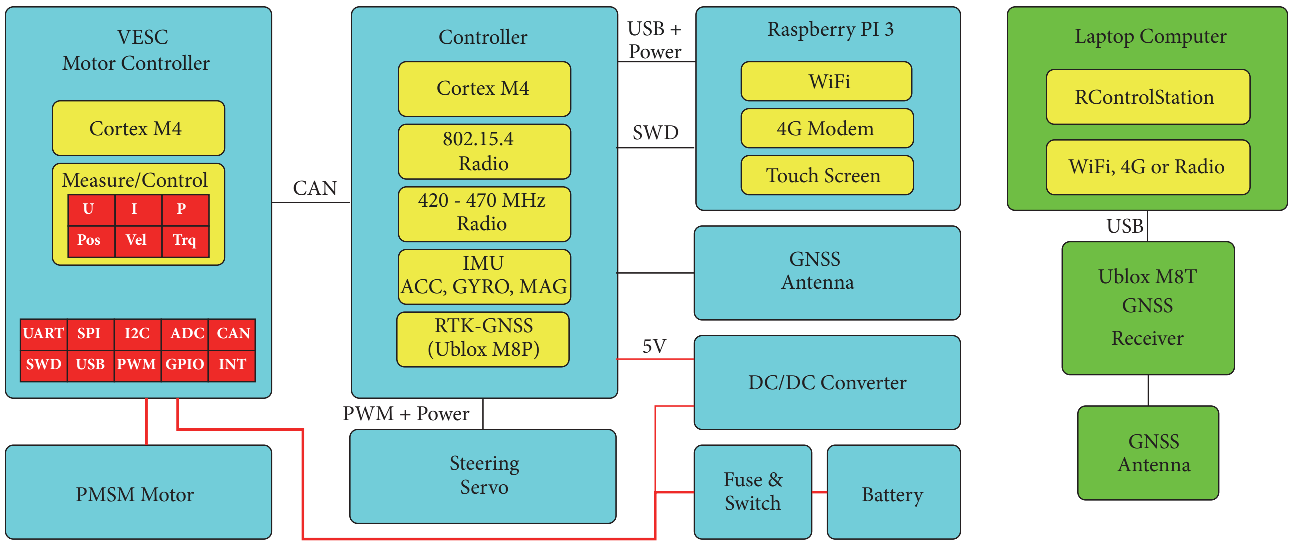


Figure 3: A block diagram of the configuration of our model car. A laptop computer for control and monitoring while acting as a RTK-SN base station is also shown.

of its configuration. Note that it also has a touch screen for the Raspberry Pi for showing network connections and other useful information, such as the battery charge level. Going through the block diagram in Figure 3, our model car, shown in Figure 2, has the following components:

(i) A lithium-ion battery with 10 cells in series and 3 Ah, providing up to 6 hours of power depending on the driving speed.

(ii) An integrated fuse and power switch, between the battery and the rest of the circuit.

(iii) A DC/DC converter that provides a 5V rail.

(iv) Our custom controller PCB, powered from the 5V rail.

(v) Our VESC motor controller, powered from the battery and connected to the controller over CAN-bus.

(vi) A PMSM motor, connected to the VESC motor controller.

(vii) A steering servo and a GNSS antenna, connected to the controller.

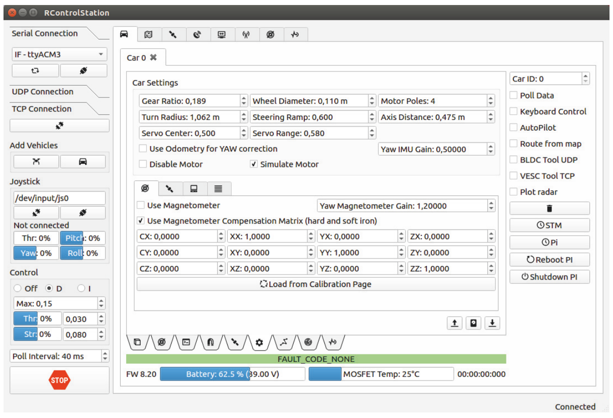
(viii) A Raspberry Pi single board computer, connected to the controller over USB. It is powered backwards through the USB ports from the controller. The Serial Wire Debug (SWD) port of the controller is also connected to the Raspberry Pi, so that the controller can be programmed and debugged remotely.

(ix) Outside the car, there is a laptop computer connected over TCP to the Raspberry Pi using the WiFi or 4G cellular connection. Our RControlStation software runs on the laptop computer and utilizes the connection to control and monitor the model car.

(x) The laptop is connected to Ublox M8T RTK-SN receiver with an antenna mounted on a tripod to act as a base station for the model car, enabling high precision positioning (see Section 3 for more details). RControlStation handles the setup of the Ublox M8T, as well as forwarding of the required correction data for RTK-SN to the model car.

The control PCB is based on an ARM Cortex M4 microcontroller and runs the ChibiOS (http://www.chibios.org) real-time operating system. The microcontroller carries out sensor fusion for position estimation (see Section 3), the trajectory following algorithm (see Section 4), and all other functionalities of the model vehicle, meaning that the connection to the laptop is only required for monitoring and sending high level control commands. In addition to the microcontroller, the control PCB contains an Inertial Measurement Unit (IMU), a CAN transceiver, two radios, DC/DC converters, an Ublox M8P GNSS receiver, and various connection ports for possible extensions.

After the control PCB, the other essential part of the electronics and software on the model car is the VESC motor controller. The VESC is also developed by us, partly in parallel with our testbed, but this paper only describes it briefly. It can drive the motor of the model car with high efficiency over a wide dynamic range using the state-of-the-art motor control technique Field Oriented Control (FOC) [12] with Space Vector Modulation (SVM) [13]. The VESC is able to operate from zero speed with high torque without position sensors on the motor by taking advantage of a nonlinear observer [14] and effective startup algorithms. Further, it provides position and speed feedback from the motor over CAN-bus as well as closed loop speed control, which is essential for the positioning system of the model car to work accurately as described in Section 3. Another essential functionality for our testbed that the VESC provides is automatic identification of all motor parameters necessary for sensorless FOC, which are rarely available in the datasheet of inexpensive model car motors. All the configuration and parameter detection of the VESC can be performed with



IF - ttyACM3

/dev/input/js0

Not connected

Thr: 0% Pitch: 0%

Yaw: 0% Roll: 0%

○ Off (●) D ○ I

Max: 0,15

Thr: 0% 0,030

Str: 0% 0,080

Poll Interval: 40 ms

Car ID: 0

Gear Ratio: 0,189 | Wheel Diameter: 0,110 m | Motor Poles: 4

Turn Radius: 1,062 m | Steering Ramp: 0,600 | Axis Distance: 0,475 m

Servo Center: 0,500 | Servo Range: 0,580

☐ Use Odometry for YAW correction | Yaw IMU Gain: 0,50000

☐ Disable Motor | ✓ Simulate Motor

☐ Use Magnetometer | Yaw Magnetometer Gain: 1,20000

✓ Use Magnetometer Compensation Matrix (hard and soft iron)

CX: 0,0000 | XX: 1,0000 | YX: 0,0000 | ZX: 0,0000

CY: 0,0000 | XY: 0,0000 | YY: 1,0000 | ZY: 0,0000

CZ: 0,0000 | XZ: 0,0000 | YZ: 0,0000 | ZZ: 1,0000

FAULT_CODE_NONE

FW 8.20 | Battery: 62.5 % (39.00 V) | MOSFET Temp: 25°C | 00:00:00:000

Figure 4: A screenshot of RControlStation, where parameters for the model car can be edited.

the accompanied VESCTool GUI, which among other things provides configuration wizards to get all settings right.

In the same way as the VESC motor controller can be configured to run with almost any PMSM without writing code and/or using expensive lab equipment for motor parameter identification, the control software can easily be configured for different model car configurations. Figure 4 shows a screenshot of RControlStation where several of the model car parameters can be edited and stored in the control PCB, such as the wheel diameter, gear ratio, and turning radius. This enables users of our testbed to configure the hardware and software to work with any model vehicle size and configuration from easy-to-use GUIs, which is one of the main objectives of our testbed. RControlStation also provides many debugging and plotting tools aiding with setting up model vehicles and performing experiments, such as magnetometer calibration and data visualization. For example, all graphs and map plots in this paper are generated in vector format using RControlStation, without requiring any additional code.

To develop and evaluate the trajectory following algorithm, as well as aiding with development of automatic test case generation with HIL, the control PCB supports a simulation mode where tests can be performed with just a USB connection to it without using the model car. This mode is implemented by simulating the behavior of the motor and mechanics with inertia and drag, and by updating the position of the vehicle with only dead reckoning from the simulated motor feedback, and heading calculated from the commanded steering angle (see Section 3 for details about the position estimation). The rate of movement of the commanded steering angle is limited to capture the behavior of a realistic steering servo. This simplifies and speeds up testing and development, while capturing many important aspects of the real-time hardware.

3. Positioning

Accurate positioning is an essential part of our testbed. Our main position source is RTK-SN, which is based on consumer GNSS technology with the addition of carrier-phase measurements of the satellite signals. The carrier-phase measurements together with conventional code measurements on both the rover (the object to be positioned) and a base station with a known position within a 10 km radius from the rover are required. This means that a data link between the rover and the base station with a data rate of around 100 bytes/second, depending on the number of satellites, is required. The data stream with code and carrier-phase measurements from the base station, together with information about the position of the base station is usually referred to as correction data and sent using a format such as RTCM3 (http://www.navipedia.net/index.php/RTK_Standards). RControlStation can either act as a base station for the model vehicles by connecting an appropriate GNSS receiver to it (such as the Ublox M8T), or act by connecting to an existing base station using the TCP or NTRIP (https://en.wikipedia.org/wiki/Networked_Transport_of_RTCM_via_Internet_Protocol) protocol. The correction data is then sent from RControlStation to the vehicles using the communication link of choice (4G, WiFi, or radio) using the RTCM3 format. The position accuracy of the rover relative to the base station with RTK-SN is around 1 cm under optimal conditions.

Traditionally RTK-SN has been an expensive technology, but recently less expensive solutions have become available [15]. In previous work we have compared the performance of high and low cost RTK-SN systems [16] and came to the conclusion that, besides the longer initial convergence time and low update rates of the low cost systems, the performance is similar. We have also studied how low cost RTK-SN performs in urban environments [17] using the same control PCB as presented here, and came to the conclusion that it performs well even when the view of the sky is poor given that multiple satellite constellations and/or sensor fusion with dead reckoning is used.

3.1. Challenges

3.1. Challenges. The main challenge with using inexpensive RTK-SN equipment with our testbed is the low position update rate, as well as latency and jitter between updates. The Ublox M8P RTK-SN receiver provides a position update rate of 5 Hz, which is too low for accurate positioning and control at speeds of up to 80 km/h, which our model cars are capable of. Figure 5 shows the measured age of consecutive position updates from the Ublox M8P. As can be seen, the samples are between 95 and 135 ms old and have jitter of up to 40 ms. If the model car moves at 10 m/s, which is less than half of the maximum speed, a latency of 100 ms causes a position error of 1 m. At the same speed, the jitter between consecutive position samples can cause errors of up to 0.5 m. Having a low update rate and high latency on one or more of the sensors on the system is a common problem within robotics, especially when low cost hardware is involved [18], and there are various methods to handle that.

To deal with the low update rate, latency, and jitter of the RTK-SN position samples, dead reckoning from sensor fusion between the odometry data from the VESC motor controller and samples from the IMU is combined with the

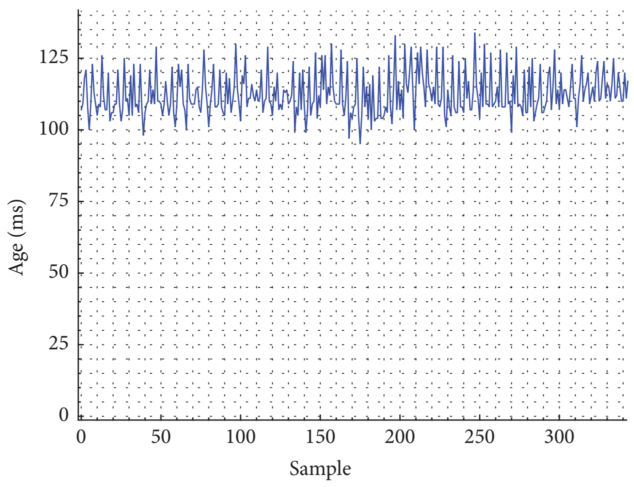


Figure 5: Delay jitter of the RTK-SN samples over time.

RTK-SN position samples to get a total update rate of 100 Hz without latency. We have come up with the method below of performing the combination. Note that the RTK-SN position in **(3)** and **(4)** first is moved to the center of the vehicle based on the current estimation of the heading angle and offset of the GNSS antenna from the vehicle center.

(1) The IMU is sampled at 1 kHz and fed to a quaternion-based orientation filter as proposed by Madgwick et al. [19], which provides the roll, pitch, and heading of the model vehicle. When the magnetometer of the IMU is configured to be included, we have improved the position filter by adding tilt compensation for estimating the heading [20] as well as ellipsoid fitting for hard and soft iron compensation of the magnetometer [21].

(2) The motor position is sampled from the VESC motor controller at 100 Hz and combined with the estimated heading from the position filter to calculate a relative dead reckoning position. This position is stored in a 100 samples long FIFO buffer together with the current GNSS time stamp derived from the pulse per second (PPS) signal from the Ublox M8P receiver.

(3) When RTK-SN position samples are received, they are compared to the dead reckoning position sample from the FIFO buffer from **(2)** with the closest time stamp to the sample, which will be 5 ms away in the worst case. The difference between the closest FIFO buffer sample and the RTK-SN sample is then used to correct the current position by moving it in the direction of the difference as

$$p_{xy} = p_{xy\_old} + d_{xy}G_{stat} + d_{xy}G_{dyn}d_p \qquad (1)$$

where p_{xy} is a vector with the new xy-position of the vehicle, p_{xy_old} is the previous position of the vehicle, d_{xy} is the difference vector between the latest RTK-SN sample and the closest position in time from the FIFO buffer, G_{stat} is a scalar configurable static gain, G_{dyn} is a scalar configurable dynamic gain, and d_p is a scalar of how far the vehicle has moved since the previous RTK-SN update. Noise between consecutive RTK-SN samples is rejected by gradually moving the current position using this method instead of moving it the full difference at once for every sample.

(4) When the magnetometer is not used to provide an absolute heading reference, the heading of the orientation filter in **(1)** is updated every time a RTK-SN sample is received by first computing an RTK-SN heading as

$$\phi_{rtk} = -a\tan 2\left(y_{rtk} - y_{prtk}, x_{rtk} - x_{prtk}\right) \qquad (2)$$

where (x_{rtk}, y_{rtk}) is the latest RTK-SN position sample and (x_{prtk}, y_{prtk}) previous RTK-SN position sample. After that a heading difference is calculated as $\phi_d = \phi_{rtk} - \phi_{FIFO}$, where ϕ_{FIFO} is the heading of the sample from the FIFO buffer from **(2)** closest to the average time between the latest and previous RTK-SN samples. The heading in the current position is then updated by rotating it in the angular direction of ϕ_d with a step proportional to how far the vehicle has moved between the latest and previous RTK-SN samples and the heading correction gain. Scaling by the distance moved is used because consecutive RTK-SN samples do not provide any heading information when the vehicle is stationary.

The rationale of this approach is that the dead reckoning position is accurate over short distances but drifts as the distance increases. The time delay of the RTK-SN samples is short enough to not cause significant degradation of the dead reckoning, meaning that the current position can be calculated accurately at a high rate by using an old absolute position and the relative movements that have occurred since then. Notice that the heading estimation is critical for the dead reckoning to perform well, and more details about that can be found in our previous work [17].

3.2. Performance Evaluation. To evaluate the performance of the latency and jitter compensation described above, we have driven the model car along a trajectory with rapid accelerations and decelerations and measured the difference between the RTK-SN position and the dead reckoning position for each sample with and without the FIFO delay compensation. Figure 6 shows the difference without compensation, and Figure 7 shows the difference with compensation. As can be seen, without the compensation the difference is stable at constant low speed but goes to 1.1 m during the accelerations and decelerations. With the compensation enabled, the difference is limited to 10 cm during the acceleration and to 20 cm during the deceleration, due to wheel slippage. The reason that the difference in Figure 6 is low during constant speed without compensation is that the position converges to an invalid value, which will be apparent when the speed changes rapidly, whereas when the compensation is enabled the position will be closer to the true position at all times. It can also be noted that during the constant speed shown in Figure 6 there is over 10 cm position jitter while Figure 7 shows less than 3 cm jitter; this is due to the delay jitter between the RTK-SN samples shown in Figure 5.

To obtain an estimate about the absolute accuracy and repeatability of the position, we have downloaded a 20 m radius circle trajectory to the model car with a constant speed of 4 km/h. While the car was following that trajectory lap after lap on artificial grass, it made visible traces that allowed us to

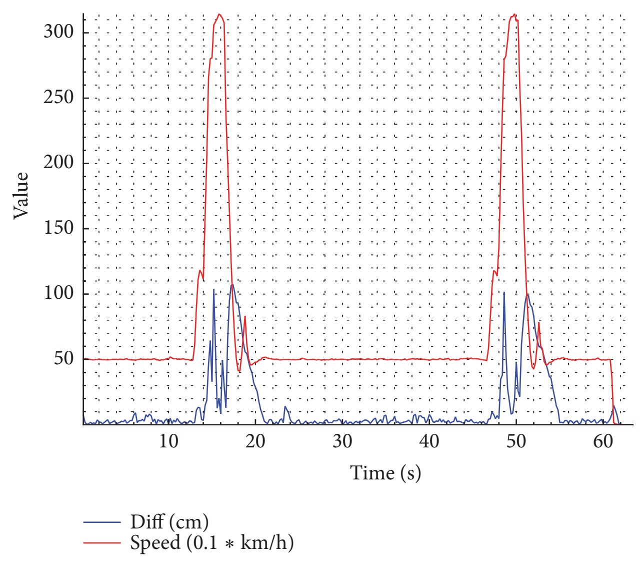


Figure 6: The speed and difference between the RTK-SN and the dead reckoning position during hard acceleration and deceleration. Delay compensation for the RTK-SN samples is disabled.

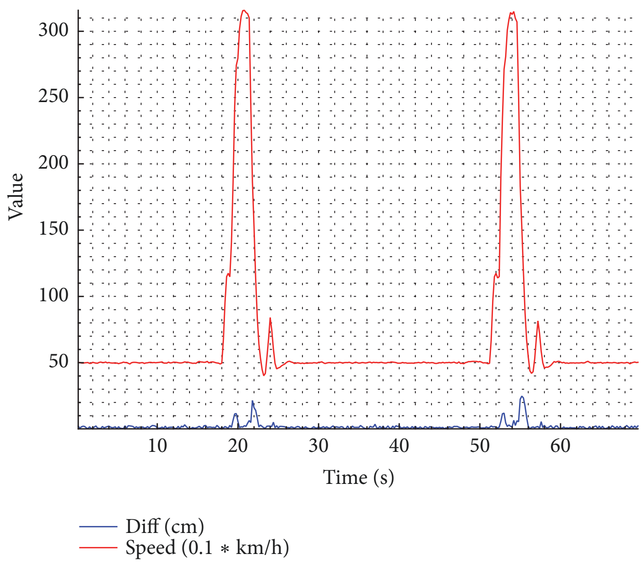


Figure 7: The speed and difference between the RTK-SN and the dead reckoning position during hard acceleration and deceleration. Delay compensation for the RTK-SN samples is enabled.

visually inspect the deviation of the car tires from the traces, as seen in Figure 8. As far as we could observe, the tires of the car stayed within the traces with less than half the tire width, or 2.5 cm, for the entire experiment. We also measured the diameter of the circle on the grass, and it had the correct diameter as accurately as we were able to measure with our equipment. The position difference between the RTK-SN samples and the closest samples in the dead reckoning FIFO stayed below 3 cm for the entire experiment, giving us confidence that our model vehicle can estimate its absolute position with 3 cm accuracy.

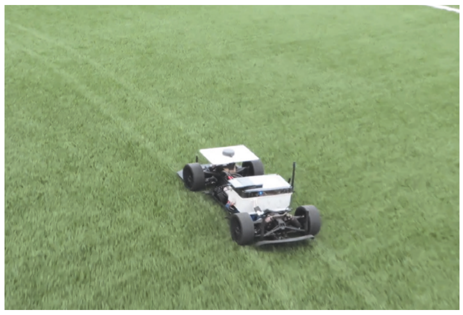


Figure 8: Our 1:6 model car repeatedly following a circular trajectory on artificial grass. The traces from previous laps are visible and can be used to estimate the lateral positioning and control repeatability.

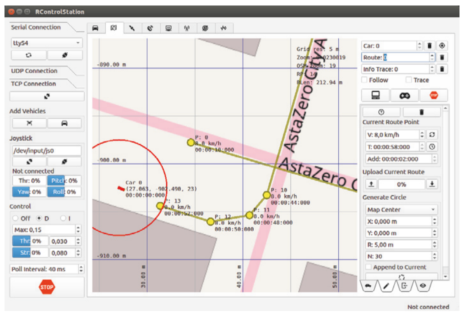


Figure 9: A screenshot of the map page in RControlStation, where trajectories can be edited.

4. Trajectory Following

An important aspect of our testbed is the ability to edit and follow trajectories. We define a trajectory as a list of points where each point has a xy-positions and a speed or time stamp, depending on the mode of operation. Our testbed support three trajectory following modes: **(1) speed based**, where the model vehicle adjusts its speed proportional to the set speed and relative distances to the two closest trajectory points; **(2) absolute time**, where the model vehicle adjusts its speed such that it reaches the trajactory points at absolute RTCM (https://en.wikipedia.org/wiki/Coordinated_Universal_Time) times (derived from the GNSS receiver clock); and **(3) relative time**, where the model vehicle adjusts its speed such that it reaches the trajectory points at times relative to when the start command was issued. Creating trajectories is most intuitive using the speed mode, but the time-based modes are necessary to synchronize multiple vehicles in a scenario. There is also a synchronization command available that can be sent to the model vehicles in real time, so that they continuously update their speed to reach a trajectory point specified in the command at the time specified in the command, based on the distance left along the trajectory to that point.

RControlStation allows users to graphically edit trajectories with an overlay of OpenStreetMap [22], as shown in the screenshot in Figure 9. We chose OpenStreetMap because it is accompanied by a complete set of tools for map creation and rendering, available as open source software. This makes it easy to update and create artificial maps, in e.g., test areas, and render them on a server. The transforms for converting OpenStreetMap rendered map tiles with Web Mercator (https://en.wikipedia.org/wiki/Web_Mercator) projection to the coordinate system of our testbed for viewing in RControlStation are well documented on their wiki page (https://wiki.openstreetmap.org/wiki/Main_Page). This was a significant aid in implementing the map rending functionality of RControlStation.

4.1. Lateral Control. Lateral control along the trajectories of the model vehicles in our testbed is performed using the pure pursuit algorithm, which is a common method within robotics [23]. Essentially it works in the following way: **(1)** draw a circle around the vehicle with a radius of the chosen look-ahead distance; **(2)** calculate the point where that circle intersects the trajectory; if multiple points are found pick the one furthest ahead on the trajectory; and **(3)** adjust the steering angle of the vehicle such that it follows an arc that intersects with that point. When visualized, it looks like the vehicle follows a point that moves away from it along the trajectory. More details about the pure pursuit algorithm can be found in the literature [23]. We have also added support for the case when the vehicle is further away from the trajectory than the look-ahead distance, in which case it will follow the closest point on the trajectory until the circle around the vehicle intersects with the trajectory, after which the algorithm is carried out as usual. This is useful when sharp turns or oscillations cause the vehicle to lose the trajectory, or when it is started far away from the trajectory. Also, if the model vehicle is to be used with automatic test case generation, it is helpful to have the ability to drive to the closest point on a recovery trajectory and drive back to the initial position along it.

Improvements to the pure pursuit algorithm commonly found in literature are interpolation of the trajectory to find points between trajectory points [23] and to use an adaptive look-ahead distance [24]. Both of these improvements have been employed in our testbed to increase the trajectory following performance. We have also implemented one to our knowledge unique improvement to the pure pursuit algorithm, which is adding gain to the steering angle calculation when the point to be followed is far away. The steering angle is corrected as

$$A_{corr} = \begin{cases} A_{st}(1 + 0.2D) & \text{if } D < 20\,\text{m} \\ 5A_{st} & \text{otherwise} \end{cases} \qquad (3)$$

where A_{corr} is the corrected steering angle, A_{st} is the steering angle that leads to the goal point along a circle that tangents with the car, and D is the distance to the goal point. The gain helps when the vehicle starts far away from the trajectory heading away from it, where it without the gain would follow an arc longer than necessary to reach the trajectory. Figure 10 shows the arc the vehicle follows without the distance gain and Figure 11 shows the arc that it follows with the gain. The starting position is the same in both figures, namely, where

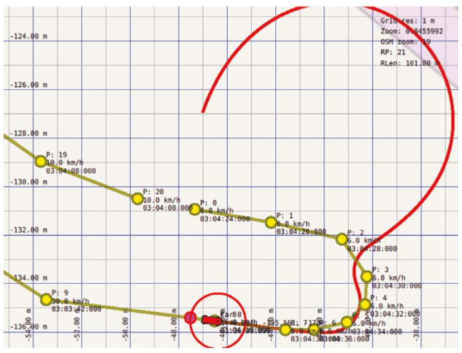


Figure 10: The red trace shows how the model car navigates back to the trajectory with angle distance gain disabled.

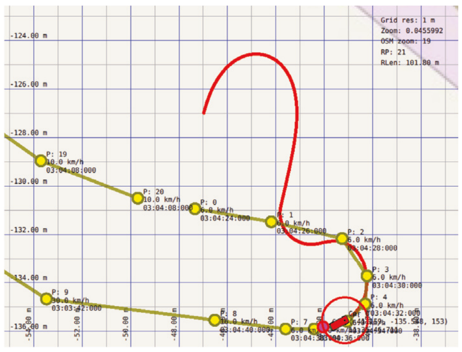


Figure 11: The red trace shows how the model car navigates back to the trajectory with angle distance gain enabled.

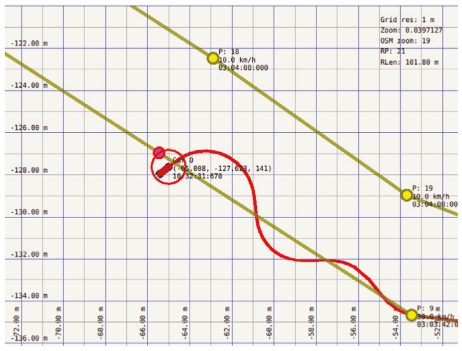


Figure 12: The red trace shows how the model car oscillates at 30 km/h with too small fixed look-ahead distance.

the red line starts. As can be seen, the arc in Figure 11 is significantly shorter. Note that the look-ahead distance of the vehicle in the figures is illustrated with a red circle around it and that the point it is aiming for is drawn on the trajectory.

Using the simulation mode of the control PCB and a trajectory that has variable speeds and sharp turns, we have set up an experiment to evaluate the performance of the lateral control with different settings and improvement strategies of the pure pursuit algorithm. First we disabled adaptive look-ahead distance and found a static distance that is long enough to follow the trajectory in a stable manner. As an unstable example, Figure 12 shows how the car behaves when the

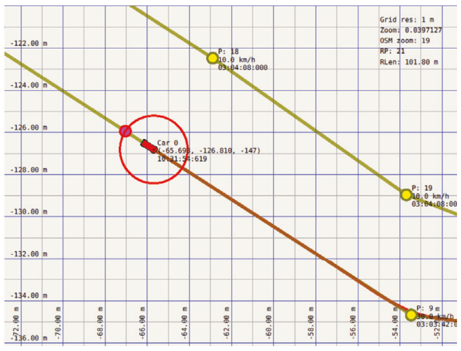


Figure 13: The red trace shows how the model car follows the trajectory at 30 km/h with sufficient look-ahead distance.

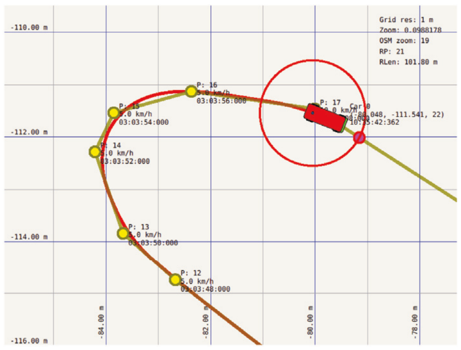


Figure 15: The red trace shows how the model car followed the trajectory with an arc with adaptive look-ahead distance enabled.

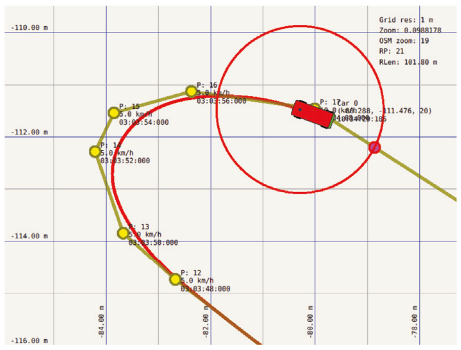


Figure 14: The red trace shows how the model car followed the trajectory with an arc without adaptive look-ahead distance enabled.

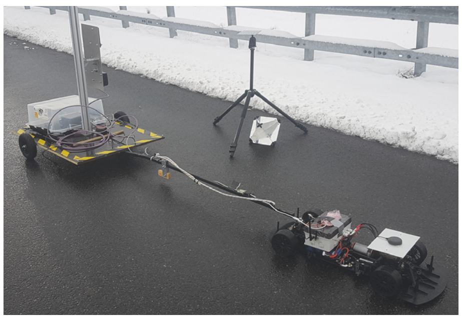


Figure 16: The model car pulling a trailer with radar to be characterized.

look-ahead distance is too short on a high speed part of the trajectory, whereas the same scenario with a sufficient look-ahead distance is shown in Figure 13. The maximum deviation from that trajectory without adaptive look-ahead distance during a stable lap was 40 cm, whereas it was 13 cm for the same track with adaptive look-ahead distance enabled. The difference in deviation from the trajectory comes from the low-speed parts of that trajectory with sharp turns, where a short look-ahead distance at low speed allows the vehicle to follow the trajectory tightly while still being stable due to the low speed. Figure 14 shows a tight turn without adaptive look-ahead distance and Figure 15 shows the same turn with adaptive look-ahead distance. For reference, the adaptive look-ahead distance is calculated based on the speed of the vehicle as

$$d = d_{base} \left(1 + |v| * 0.05\right)^2 \tag{4}$$

where d is the calculated adaptive look-ahead distance, v is the speed in m/s, and d_{base} is the look-ahead distance when the speed is 0. This equation was derived experimentally.

5. Conclusion

In this paper we have presented a novel testbed for research and development within the areas of autonomous driving and accurate positioning. We were able to achieve high position accuracy and low latency using low-cost hardware by fusing data from multiple sensors (Accelerometer, Gyroscope, odometry, and RTK-SN) to take advantages of their individual strengths. For trajectory following we have implemented the well-known pure pursuit algorithm along with two common and one unique improvements as described in Section 4.

Model vehicles based on our testbed can follow trajectories autonomously, and all tools necessary for visualization and trajectory generation are provided. Our testbed is scalable to a wide range of model vehicles and can be set up in just one day of work given familiarity with the testbed. The custom hardware and all involved software are open source for easy development and extension. We have also performed a wide range of tests to ensure high performance and reliability in various situations. To our knowledge there is no similar testbed available today, and there is a significant need for it in the research and education communities.

In addition to use within research and education, our testbed can be used in data acquisition applications where sensors need to be moved accurately according to a defined path, while data is stored together with position and speed. For example, Figure 16 shows a photo of our model car pulling a trailer with radar to be characterized along a predefined path around different targets. The open source nature of our testbed provides a solid base for such applications and enables further functionality to be implemented without the burden of implementing the positioning and navigation functionality.

Glossary

CAN:	Controller area network. A communication bus standard for industrial and automotive applications.
Dead reckoning:	The process of estimating the current position by advancing a previously determined position using inertial or speed measurements. Dead reckoning generally relies on measuring derivatives, meaning that the error will drift without an upper bound if no absolute measurements, e.g., GNSS, are used
FOC:	Field Oriented Control (see [12])
GNSS:	Global Navigation Satellite System
GUI:	Graphical User Interface
HIL:	Hardware-In-the-Loop
IMU:	Inertial Measurement Unit. Achip that can measure acceleration and angular velocity with three axes each
MPC:	Model predictive control (see [2])
Odometry:	Tracking the rotation of the wheels. In our case this is done by tracking the rotation of the motor
PCB:	Printed circuit board
PMSM:	Permanent magnet synchronous motor. A three-phase motor without brushes, often referred to as BLDC motor
PPS:	Pulse per second. A pulse that is emitted with accurate timing in the beginning of each second, synchronized to GNSS time
RTK-SN:	Real-time kinematic satellite navigation (see [15, 16])
SVM:	Space vector modulation (see [13])
SWD:	Serial wire debug. A programming and debugging interface.

Data Availability

As our testbed is open source, including all PCB designs, embedded software, and desktop software, the experiments presented in this paper can be replicated by constructing a model car using the resources available at https://github.com/vedderb/rise_sdvp. The data, plots, and trajectories, as well as additional data and plots from the experiments presented in this paper, can be found here https://github.com/vedderb/rise_sdvp/tree/master/Misc/Test%20Data/paper_2018-06. To replicate the experiments and collect similar data, as well as to generate similar plots, the trajectories in the routes directory from the previous link can be used together with the appropriate terminal commands from the car terminal of RControlStation. RControlStation is part of the open source software, and all terminal commands can be listed by typing help in the car terminal.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research has been funded through EISIGS (grants from the Knowledge Foundation) and by VINNOVA via the FFI Projects Chronos steps 1 and 2.

References

[1] F. C. Braescu and C. F. Caruntu, “Prototype model car design for vehicle platooning,” in *Proceedings of the 2nd International Conference on Optimization of Electrical and Electronic Equipment, OPTIM 2017 and Intl Aegean Conference on Electrical Machines and Power Electronics, ACEMP 2017*, pp. 953–958, Brasov, Romania, May 2017.

[2] F. C. Braescu, “Basic control algorithms for vehicle platooning prototype model car,” in *Proceedings of the 21st International Conference on System Theory, Control and Computing, ICSTCC 2017*, pp. 180–185, Sinaia, Romania, October 2017.

[3] A. Fenesan, D. Szöcs, T. Pana, and W-H. Chen, “Building an electric model vehicle and implementing an obstacle avoidance algorithm,” in *Proceedings of the 2012 International Conference and Exposition on Electrical and Power Engineering (EPE)*, pp. 49–53, Iasi, Romania, October 2012.

[4] F. Bormann, E. Braune, and M. Spitzner, “The C2000 autonomous model car,” in *Proceedings of the 4th European DSP Education and Research Conference, EDERC 2010*, pp. 200–204, France, December 2010.

[5] J. Axelsson, A. Kobetski, Z. Ni, S. Zhang, and E. Johansson, “MOPED: A mobile open platform for experimental design of cyber-physical systems,” in *Proceedings of the 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, pp. 423–430, Italy, August 2014.

[6] G. Halfacree and E. Upton, *Raspberry Pi User Guide*, Wiley, 1st edition, 2012.

[7] R. Svenningsson, R. Johansson, T. Arts, and U. Norell, “Formal Methods Based Acceptance Testing for AUTOSAR Exchangeability,” *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 5, no. 1, pp. 209–213, 2012.

[8] M. Pahlavan, M. Papatriantafilou, and E. M. Schiller, “Gulliver: A Test-Bed for Developing, Demonstrating and Prototyping Vehicular Systems,” in *Proceedings of the 2012 IEEE Vehicular Technology Conference (VTC 2012-Spring)*, pp. 1-2, Yokohama, Japan, May 2012.

[9] K. Hulme, E. Kasprzak, K. English, D. Moore-Russo, and K. Lewis, “Experiential learning in vehicle dynamics education via motion simulation and interactive gaming,” *International Journal of Computer Games Technology*, vol. 2009, Article ID 952524, 15 pages, 2009.

[10] A. Soltani and F. Assadian, “A hardware-in-the-loop facility for integrated vehicle dynamics control system design and validation,” *IFAC-PapersOnLine*, vol. 49, no. 21, pp. 32–38, 2016.

[11] A. Mouzakitis, D. Copp, R. Parker, and K. Burnham, “Hardwarefor testing automotive ECU diagnostic software,” *SAGE Measurement and Control Journal*, vol. 42, no. 8, pp. 238–245, 2009.

[12] J. P. John, S. S. Kumar, and B. Jaya, "Space vector modulation based field oriented control scheme for brushless DC motors," in *Proceedings of the 2011 International Conference on Emerging Trends in Electrical and Computer Technology, ICETECT 2011*, pp. 346–351, India, March 2011.

[13] M. Gaballah and M. El-Bardini, "Low cost digital signal generation for driving space vector PWM inverter," *Ain Shams Engineering Journal*, vol. 4, no. 4, pp. 763–774, 2013.

[14] J. Lee, J. Hong, K. Nam, R. Ortega, L. Praly, and A. Astolfi, "Sensorless control of surface-mount permanent-magnet synchronous motors based on a nonlinear observer," *IEEE Transactions on Power Electronics*, vol. 25, no. 2, pp. 290–297, 2010.

[15] T. Takasu and A. Yasuda, "Development of the low-cost RTK-GPS receiver with an open source program package RTKLIB," in *Proceedings of the International Symposium on GPS/GNSS*, pp. 4–6, Jeju, Korea, 2009.

[16] M. Skoglund, T. Petig, B. Vedder, H. Eriksson, and E. M. Schiller, "Static and dynamic performance evaluation of low-cost RTK GPS receivers," in *Proceedings of the 2016 IEEE Intelligent Vehicles Symposium, IV 2016*, pp. 16–19, Sweden, June 2016.

[17] B. Vedder, J. Vinter, and M. Jonsson, "Accurate positioning of bicycles for improved safety," in *Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, Las Vegas, Nev, USA, January 2018.

[18] M. Bošnak, D. Matko, and S. Blažič, "Quadrocopter hovering using position-estimation information from inertial sensors and a high-delay video system," *Journal of Intelligent Robotic Systems*, vol. 67, no. 1, pp. 43–60, 2012.

[19] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," in *Proceedings of the IEEE International Conference on Rehabilitation Robotics (ICORR '11)*, pp. 1–7, July 2011.

[20] T. Ozyagcilar, "Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors, An4248," Freescale Semiconductor, Application Note, 2015.

[21] A. Vitali, "Ellipsoid or sphere fitting for sensor calibration, Dt0059," ST Microelectronics, Design Tip, 2016.

[22] M. Haklay and P. Weber, "Openstreetmap: user-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[23] H. Ohta, N. Akai, E. Takeuchi, S. Kato, and M. Edahiro, "Pure pursuit revisited: Field testing of autonomous vehicles in urban areas," in *Proceedings of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA 2016*, pp. 7–12, Japan, October 2016.

[24] M. Park, S. Lee, and W. Han, "Development of lateral control system for autonomous vehicle based on adaptive pure pursuit algorithm," in *Proceedings of the 2014 14th International Conference on Control, Automation and Systems (ICCAS)*, pp. 1443–1447, Gyeonggi-do, South Korea, October 2014.