ENVIRONMENT SURVEYING AUTOMATED VEHICLE

HAMK
HÄMEEN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

SICK
Sensor Intelligence.

SUOMI
GENIUSES

Khoa Dang
Silvia Lupea
Samuli Multaniemi
Giang Nguyen
Minh Tran

Table of contents

Figures and Tables

Acknowledgements

We, Suomi Geniuses, would like to send our heartfelt thanks to SICK Finland for providing us the equipment and documentation for realizing this innovation.

Also, to HAMK University of Applied Sciences for facilitating and supporting us in this project. Especially, we want to express our gratitude to:

- Ms. Katariina Penttilä
- Mr. Jan-Peter Nowak
- Mr. Juhani Henttonen
- Mr. Juha Sarkula

for providing their support and expertise, without which we could not have finished this project.

Finally, we want to thank our family for all their love and support for us.

1 Introduction

In Finland, over 55 thousand automobiles traverse on approximately 140,000 kilometers of road each year, according to the Finnish Transport Agency. The need for road condition monitoring and maintenance is a legitimate concern for the people and the governing body, as the road quality is a major factor deciding the safety of travellers. At the same time, economic efficiency depends greatly on the road quality, as it relates directly to the fuel and transportation efficiency.

Road constructions companies, namely Destia, has specific equipment for profiling the road surface and provide data such as road unevenness, wheel groove depth and texture. The limitation of these devices is the lack of surrounding data, as a profiling sensor can only survey the parallel plane (less than 180-degree aperture), whereas a laser scanner, in most cases, can acquire data in a 270-degree field, or in some cases 360 degree. Moreover, the measurement range of a laser scanner is far superior compared to a profile measurement device. The only drawback of using a laser scanner is the inconsistency of horizontal resolution, which can be easily overcome by adding additional scanners on the horizontal axis. In addition to this, the price per unit of a laser scanner is 5 to 10 times less than a profile measurement device. For road and environment terrain surveying, the resolution of a laser scanner is adequate. In general, for terrain mapping applications, theoretically a laser scanner performs better economically compared to a profiling sensor, while still maintaining the required accuracy for said applications.

This report documents the design, functionality description and presents areas for improvement of the Environment Surveying Automated Vehicle (ESAV). This is a proof-of-concept that was developed to prove that the laser scanner can replace surface profiling sensors in terrain mapping applications.

To correctly reconstruct the environment terrain, the orientation angles of the laser scanner and the moving speed of the scanner had to be known. The solution for collecting environmental data was to mount the laser scanner on a vehicle, facing down to the ground. In details, the solution consisted of three parts:

- An automated car, powered by the Arduino Mega 2560 microcontroller. Features of the car included speed control and steering using differential two-wheel drive. A 200 Hz Inertial Measurement Unit (IMU) sensor fusion algorithm was implemented to provide orientation angles of the car.
- A Raspberry Pi 3 for interfacing with the SICK TiM561 Laser Scanner. The Pi communicates with the Arduino via a 2Mbps serial connection to acquire speed and orientation angles, and to give the speed setpoint for the car.
- An offline visualization software to apply post-processing to the scan data and render the scanned surface.

2 Problem statement – Solution overview

To achieve the target of the project, the following problems must be solved:

1. Redesign and renovate the mechanical structure of the car, which was reused from a previous competition by another team of students from HAMK
2. Control the car to run at a specific speed and steering, if possible with the current car
3. Get the orientation angles of the car for applying correction to the scan data
4. Interface with the TiM561 laser scanner and store the scan data for use later
5. Apply data correction to the scan sample and render the scan data on the visualization program

The second and third problems deal with embedded programming, whereas the remaining problems deal with signal processing and using different APIs to realize the objectives. In general, the following solutions were derived to reach the project goal:

- Car velocity control using a fuzzy PI-controller.
- Reading accelerometer, gyroscope and magnetic compass sensor data, applying data fusion using Mahony's fusion algorithm to estimate the orientation angles of the car and the laser scanner.
- SGPiApp, a C++ application running on the Raspberry Pi 3, written to achieve the following goals:
    - Remote control of the car via SSH or TeamViewer
    - Give velocity setpoint to the Arduino board, via Serial communication
    - Communicate with the TiM561 and store the measurement data
- SGApp, a C++ application running on a Windows or Linux computer, written to achieve the following goals:
    - Apply Savitzky-Golay filtering algorithm to the scan data to reduce measurement noise
    - Render the scan data after correction

The following video demonstrates the operation of the car:
https://www.youtube.com/watch?v=cIrN72iAXGk

2.1. Mechanical renovation of the existing car

The timeline of this project was limited for a full design schedule and the vehicle used in our application is more of a housing and an automation solution instead of the aspect focused onto in the project. Therefore, the base for the physical design was an autonomous robot car used in a field robot competition a few years before made also by our school's automation students.

After preliminary inspection, the conclusion was reached that most of the design had to be redone as their model of operation was very different and had components divided onto a trailer and the actual chassis itself. It was decided to dismantle everything from the aluminum frame and started to work from an empty frame. The renovation process was quite lengthy and consisted of straightening the 4 motor axis rods, removing unnecessary brackets and structure elements, adding our own component, junction box rack rails and removing traction improving studs from the tires, as they were making the ride too bumpy when used with the hard-vulcanized rubber tires. Our components also needed housing and wiring, and we wanted to make them as convenient as possible. We divided our components into two junction boxes and inserted plenty of cable glands, so that cables won't be too tight anywhere. We wanted to treat the physical implementation of this project as a proof of concept for our designed application, the environment mapping.

In conclusion, we chose for a functionality over appearances –approach. Another key element in the design was modularity, and the placement of the components were chosen to allow easy accessibility and swapping.

2.2. Velocity control and IMU fusion algorithm

In this project, an Arduino Mega 2560 board (utilizing a ATMEGA2560 microcontroller) was used to achieve car velocity control and carry out the sensor fusion algorithm to determine the orientation angles of the car (also known as attitude). To program the Arduino board, usually either the Wiring platform or Atmel's C++ platform are used. The main characteristics of the two programming platforms is described as:

- Arduino Wiring platform:
  - Abstract and simple interface for common features of a microcontroller.
  - Initialization and simple usage of some microcontroller peripherals, for example I^2C bus or serial communication.
  - Lack of optimization and customizability, compared to a traditional microcontroller programming language.
  - Allows integration of Atmel's C++ into the code.

- Atmel’s C++ programming:
    - Fully customizable in terms of peripherals usage (timers, counters, interrupts, etc.).
    - Requires in-depth knowledge of embedded programming. For example, initialization of peripherals can be complex for a non-embedded programmer.
    - Compiler optimization level can be customizable to favor program size, program speed or a balance of the two.

The embedded program was a mix of Arduino Wiring and Atmel’s C++ microcontroller programming, to achieve the balance between ease of programming and functionality performance. In general, the embedded programming has the following features:

- Interrupt-based encoder sampling, rotational speed calculated every 30 milliseconds.
- Velocity control using a primitive fuzzy PI controller.
- Mahony’s sensor fusion algorithm for the IMU, running at 200 Hz.
- Serial communication at 2 million bits per seconds (2 Mbps)

The programming was made possible thanks to the open-sourced non-licensed libraries provided by the Arduino programmer community. The libraries and their usage were as followed:

- Encoder library by Paul Stoffregen: https://github.com/PaulStoffregen/Encoder - Used without modification
- PinChangeInterrupt library by GreyGnome: https://github.com/GreyGnome/PinChangeInt - Used without modification
- TimerThree library by Paul Stoffregen: https://github.com/PaulStoffregen/TimerThree - Used without modification. TimerFour and TimerFive libraries were written based on the TimerThree library and the ATMEGA2560 microcontroller datasheet.
- MPU-9250 example sketch by kriswiner: https://github.com/kriswiner/MPU-9250 - Modified for operation based on interrupt, program size, constant coefficients and more optimized variable usage.

2.2.1. Velocity control

Velocity control of the car is being done indirectly through controlling the rotational speed of the wheel motor. The Arduino receives a speed setpoint value, in millimeters per second, command from the Raspberry Pi 3 and returns the current velocity, measured via multiplying the wheel rotational speed with the wheel radius, and the orientation angles to the Pi.

A primitive fuzzy PI controller was implemented for execution speed reasons, measuring the rotational speed of the wheel in radians per second and vary the pulse width modulation (PWM) duty cycle to keep the rotational speed as close to the setpoint as possible. The PWM frequency was set at 20 kHz, to avoid unwanted noise to the end user. Although the driver module datasheet stated the ability to operate at 25 kHz switching frequency, experiments have shown a decrease in efficiency and an increase in heat dissipation, compared to a lower switching frequency such as 20 kHz.

The encoder modules used were manufactured manually, therefore they did not have a high pulses-per-revolution (PPM) count (178 PPR) and the A-B output phase difference was not exactly 90 degrees. Anyhow, the results were adequate for measuring the running direction and speed at 33 Hz. The motors were driven using two IBT-2 H-bridge modules, each driving the motors on one side. The module employed two BTS7960 half-bridge IC, capable of driving loads up to 43 amperes at the maximum switching frequency of 25 kHz.

Three high-resolution timers and four external interrupts in the ATMEGA2560 microcontroller was used to achieve this task. The four external interrupts were used for the encoders output; timer 3 was used for scheduling the velocity calculation and speed control subroutine; and timers 4 and 5 was used to generate the 20 kHz PWM signals for driving the motors.

2.2.2. IMU data fusion algorithm

The inertial measurement unit used in this project was Invensense MPU-9250, which contains an Invensense gyroscope – accelerometer and a third-party magnetometer. The unit was configured to sample acceleration and rotational speed at 1000 Hz, averaged down to 200 Hz to match with the sampling frequency of the magnetometer. The module communicates with the Arduino board via 400 kHz I^{2}C bus and uses an interrupt pin to signal the Arduino board to start the fusion subroutine. Figure 1 illustrates the orientation angle of a rigid body in 3D space.

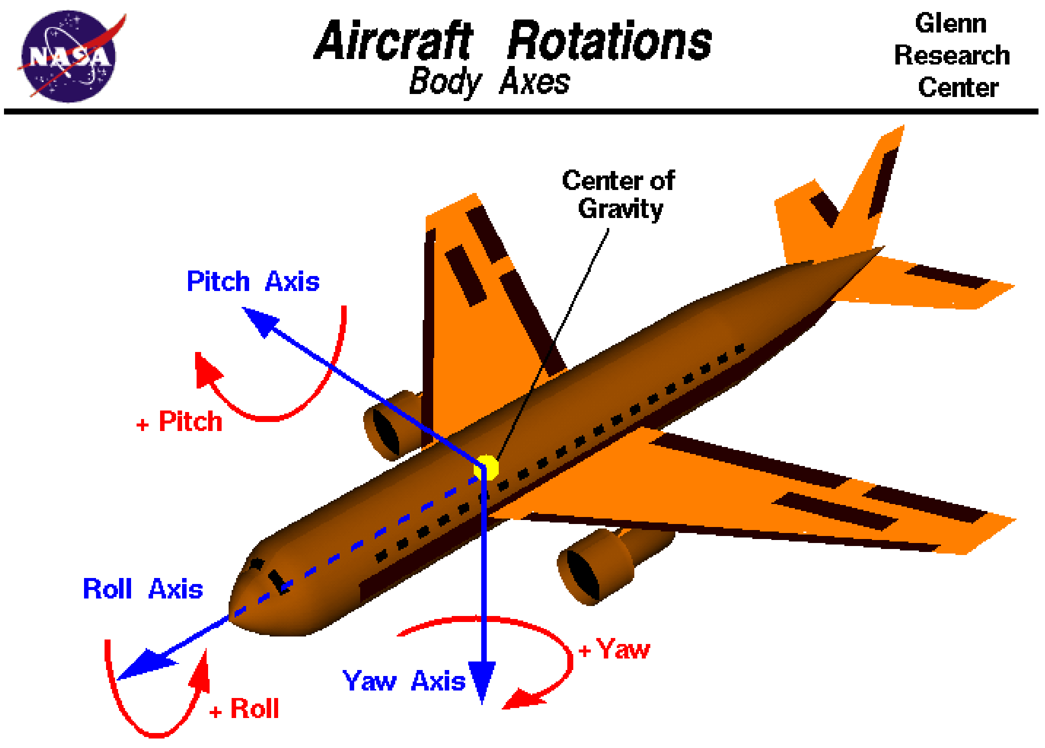

Figure 1 Orientation angles (NASA, 2015)

According to Abyarjoo in "Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magnetic Sensors" (2012), the measurement basis for the fusion algorithm can be summarized as following:

- Integrate the gyroscope measurements (ω_x, ω_y, ω_z) over time to get angular displacement.
    - This method suffers from the gyroscope measurement white noise, which leads to measured angle drift over time.

- Use the acceleration measurements (a_x, a_y, a_z) and calculate the angles between them in respective planes to acquire the orientation angles.
    - The measurement noise in this measurement results in noisy angle results, although the measured values suffered no drifting.
- Use the magnetometer measurements (m_x, m_y, m_z) to measure the heading angle.
    - The magnetometer, in most cases, is slower compared to the other two sensors and provides only the yaw angle.

Mahony's fusion algorithm tries to estimate the earth-frame gravity vector, g_{earth} and the gravity vector in the sensor's body-frame, g_{body}. By determining the rotation matrix that turns g_{earth} into g_{body}, the orientation angles of the object can be derived. Mahony used quaternions to represent the rotation matrix, as this method is less computationally expensive compared to the Euclidean representation.

The orientation of the sensor in 3D space at time *n* is represented as the state quaternion $q[n]$. The rate of change of this quaternion, denoted as $\dot{q}[n]$ is the quaternion product of the previous state quaternion $q[n-1]$ and the corrected angular velocity $\Omega[n]$:

$$q[n-1] = [q_1 \quad q_2 \quad q_3 \quad q_4]$$
$$\dot{q}[n] = \frac{1}{2}\, q[n-1] \otimes \Omega[n] = \frac{1}{2}\, q[n-1] \otimes [0 \quad \overline{\omega}_x \quad \overline{\omega}_y \quad \overline{\omega}_z]$$
$$q[n] = q[n-1] + \dot{q}[n] * T_S$$

The quantity T_S is the sample period of the system, which is 5 milliseconds.

The state quaternion was used to derive g_{body}; whereas the accelerometer and magnetometer measurements were used to estimate g_{earth}. The difference between g_{body} and g_{earth}, which can be understood as the "error", is then used as a feedback term to correct the gyroscope measurements before integration, using a PI-controller, to derive the term $\Omega[n]$. From the state quaternion, the orientation angles could be derived as following:

$$yaw = atan2(2q_2q_3 - 2q_1q_4\, , 2q_1^2 + 2q_2^2 - 1)$$
$$pitch = -\sin^{-1}(2q_2q_4 + 2q_1q_3)$$
$$roll = atan2(2q_3q_4 - 2q_1q_2\, , 2q_1^2 + 2q_4^2 - 1)$$

For this project, Madgwick's implementation of Mahony's fusion algorithm was used to estimate the orientation of the car from the measurement values of the MPU-9250 sensor.

2.3. SGPiApp – Raspberry Pi 3 onboard program

To collect scan data from SICK scanner and various motion information of the vehicle to the Raspberry Pi computer, a program - SGPiApp - is built to run on the Raspberry Pi. This program acquires scan data from the SICK.

The program was designed to be multi-threaded to take advantage of the multicore structure of the Raspberry Pi 3, aiming for faster operations and integration of a computer vision system later. After reconsideration, the vision system was not implemented due to the lack of training samples and performance.

2.3.1. Platforms and APIs

Targeted operating system is Raspbian Jessie, which is based on Debian Linux. The source code had to be compiled and built on the system although the programming and testing process was done on another computer running Ubuntu Linux. No obstacle was encountered while building the program on the Raspberry Pi since the systems are quite similar.

To build the source code on Raspberry Pi, it needs to be remotely accessed from another computer over the internet by using either Secure Shell (SSH) program or TeamViewer software.

The program is developed using C++ programming language, complied by the GNU C++ compiler. The application programming interfaces (APIs) utilized in the code includes:

- libusb for interfacing the SICK TiM561 with the Raspberry Pi
- pthread for multi-threading the program
- RS232 for serial communication with the Arduino

2.3.2. Car motion data acquisition

Motion data for the car included velocity and yaw-pitch-roll angles. Yaw-pitch-roll angles were calculated in the Arduino using the information acquired from the inertial measurement unit. All these four variables are floating point value type (4 byte). Arduino is connected to Raspberry Pi via an USB interface, for power and 2Mbps serial communication. SGPiApp program uses RS232 API to make serial communications (reading and writing) with the Arduino. Every 33ms, SGPiApp sends a request command, containing a character ‘R’ and the velocity set point at which the car is expected to run. Then Arduino replies by sending the yaw-pitch-roll angles together with the current velocity measurement and a confirmation code value (0xFA – hexadecimal value).

At the beginning of the session, SGPiApp is started by an operator. The program reads the command-line for the velocity set point from the operator. At initialization phase, serial communication with the Arduino is started. The program waits until the Arduino program has finished initializing and sends back an initialization confirmation code (0xAF – hexadecimal value). The program then requests for the first packet of information for checking. If the values of any of the rotation angles is not calibrated and shows a biased value compared to zero, the serial communication and Arduino is reset until the returned angle values are zeroes. The initialization process finishes when the first data packet from the Arduino shows reasonable values for the yaw pitch and roll. The communication with Arduino is terminated when the program stops.

2.3.3. Scanner data acquisition:

Scanner data is acquired from the SICK TiM561 scanner over the USB interface connecting the scan device and the Raspberry Pi. To receive scan data, the program first initializes the configuration parameters for the sensor and triggers the streaming of data packs. Each data pack sent by the scanner during run time contains information of one scan as is specified in the device’s telegrams manual. The program parses each telegram and produces an equivalent scan data pack structure with reduced information from the telegram and combined motion data acquired from the car. Table 1 shows the scan data pack structure used in the program.

Field name		**Description**	**Data type**	**Size (bytes)**
Num_points		Number of points in one scan	uint32	4
Time_stamp	Hour	The time at which the scan was taken	uint32	4
	Minute		uint8	1
	Second		uint8	1
	Millisecond		uint8	1
Velocity		Velocity of the car when the scan was taken	float	4
Rotation angles	Yaw	Yaw, pitch and roll angles taken from the inertial measurement device when the scan was taken	float	4
	Pitch		float	4
	Roll		float	4
Range list		A vector containing the ranges or the distance between each data point in the scan and the sensor	float array	4*Num_points

Table 1 Scan data pack data structure

Note: uint32 – unsigned integer 32 bit, uint8 – unsigned integer 8 bit

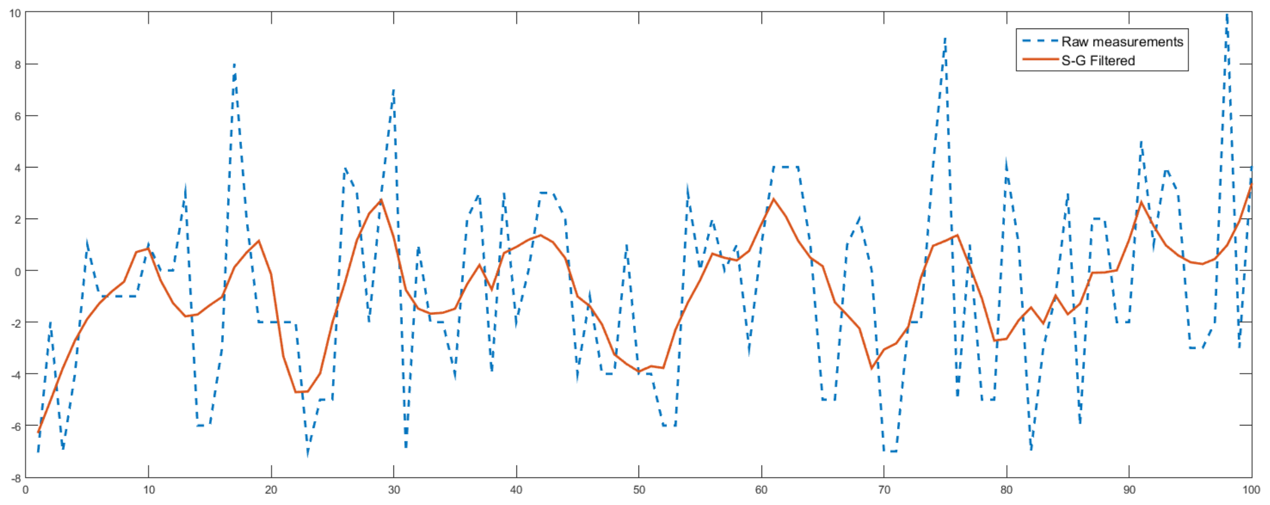

Figure 2 Scan range of one point, over 100 samples

Figure 2 illustrates the measurement of the TiM561 on a leveled surface, 251 mm away from the scanner, at 90-degree angle. The raw value is then subtracted from 251 to get the altitude measurements. The “true” measured value should be zero, but the measured values varied between -7 to 10 mm, therefore an accuracy of ± 15mm was assumed for the scanner. The calculated standard deviation of the sample was 3.84 mm and after applying the Savitzky – Golay filter, the deviation dropped down to 1.93 mm and it can be clearly seen that the measurement accuracy was improved.

2.3.4. Road record

For each running session of the SGPiApp, a road record is created which stores the configuration information for the scanner and the data collected from the sensors. The program stores all scan data packs in a vector to make the data content for the road record. Table 2 demonstrates the data format of a road record file. The road record is stored on the SD card of the Raspberry Pi in folder “out/” of the same folder with the SGPiApp program file.

To deal with memory overhead during the recording of the road record, after a fixed interval during run-time, the program flushes the data recorded out to a backup record file and frees the memory used so far. By the end of the program, the backup files are combined to produce a complete road record file.

The binary files containing a road record follow the format described in table 2. File name extension is .dat, for example “record_file.dat”.

Field name		**Description**	**Data type**	**Size (bytes)**
Num_packs		Number of packs in a record	uint32	4
Start time	Year	The time at which the program starts recording the session	uint32	4
	Month		uint8	1
	Day		uint8	1
	Hour		uint8	1
	Minute		uint8	1
	Second		uint8	1
Start angle		Starting angle for each 2D scan	float	4
End angle		Ending angle for each 2D scan	float	4
Angular step		The angle resolution of each scan	float	4
Data		A list of data scan packs	Scan data structure (see Table 1)	(Size of scan data structure) * Num_packs

Table 2 Road record data structure

2.4. SGApp – Data processing and visualization program

SGApp is a desktop application developed to process the recorded data from the vehicle and visualize the data after processing on the screen.

2.4.1. Platforms and APIs

The application target operating systems are Windows and Linux. wxWidgets library is used to build a graphical window, while the OpenGL APIs is used for 3D visualization of the recorded data.

2.4.2. Data post-processing

Savitzky-Golay filter was used for data post-processing. The filter is known for improving the Signal-to-Noise ratio, which reduces measurement noise from the scan data while preserving the shape and time characteristics.

The filter is applied to the following data:

- Roll, pitch and yaw angles
- Velocity
- Scan data of the same angle in the dataset.

2.4.3. Scan data point correction

The program used a vector to store the coordinates of each scan data points. Every time a new record file is loaded for visualization, this vector is reinitialized.

Before rendering each point in the world coordinates, the positions of each data points need to be calculated in combination with changes of yaw-pitch-roll angles of the vehicle by applying geometry mathematics.

Transformation matrices:

Before explaining how the points are corrected in world coordinates, the mathematics of transformation matrices are demonstrated hereby.

The rotation matrices around the x-, y- and z- axis with rotation angle alpha, beta and gamma respectively in the 3D space are shown in figure 3.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\ \alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3 Rotation matrices

Translation transformation by a vector v in 3D space is done using the translation matrix as shown in figure 4.

$$T_V = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4 Translation matrix

To apply a sequence of transformations, individual transformation matrices are multiplied in reversed order.

First, the matrix representing the coordinates of the scan points acquired from the sensor in each data pack in the XY plane were calculated, with the SICK sensor being the origin of the plane. Then translation and rotation matrices are applied to find an equivalent transformation matrix for the SICK sensor in world coordinates. Finally, the two matrices are multiplied to help find the world coordinates of each scan data point. The way to apply these transformations to the scanner data are explained with more details in the following sections.

Points coordinates with scanner device as origin:

The coordinates of each point in the XY plane, with the 2D scanner at the center, are calculated by the following formulas:

$$x_i = range_i * \cos(alpha_i)$$
$$y_i = range_i * \sin(alpha_i)$$

$$Where: alpha_i = start\ angle + angle\ resolution * i$$

$$0 \leq i < number\ of\ points$$

Figure 5 illustrates the scan data of a rectangular box on a leveled surface, when the scanner is tilted.

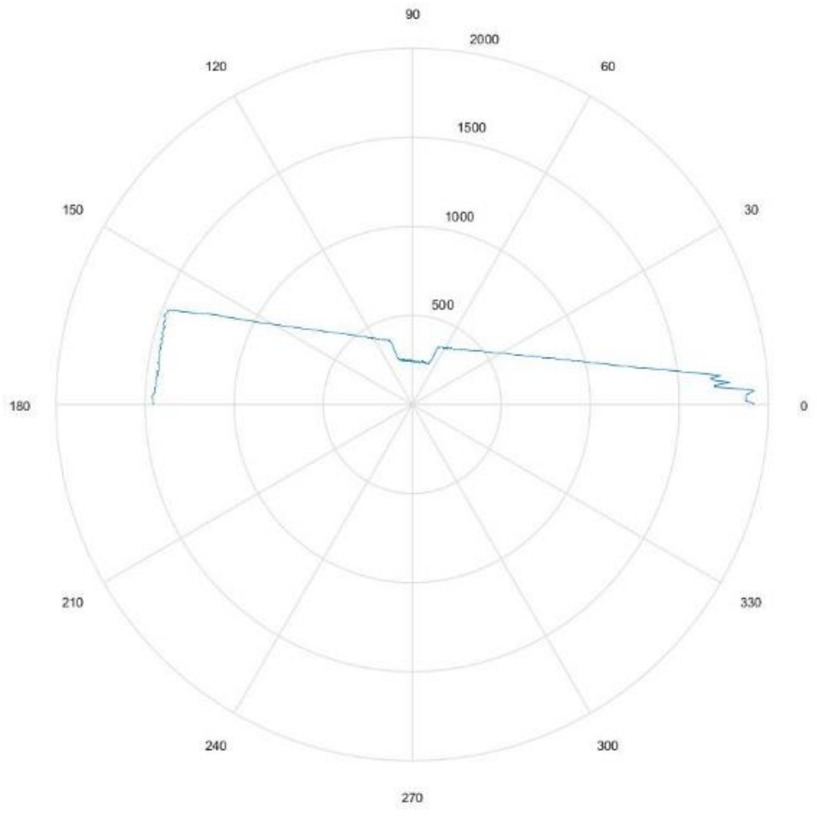

Figure 5 Scan data points in X-Y plane

The coordinates of each point are put in a matrix representing a translation transformation, from the scanner point.

SICK scanner transformation in world coordinates:

At balance, where the roll or pitch changes are negligible, the scanner faces the world origin as displayed in figure 6.

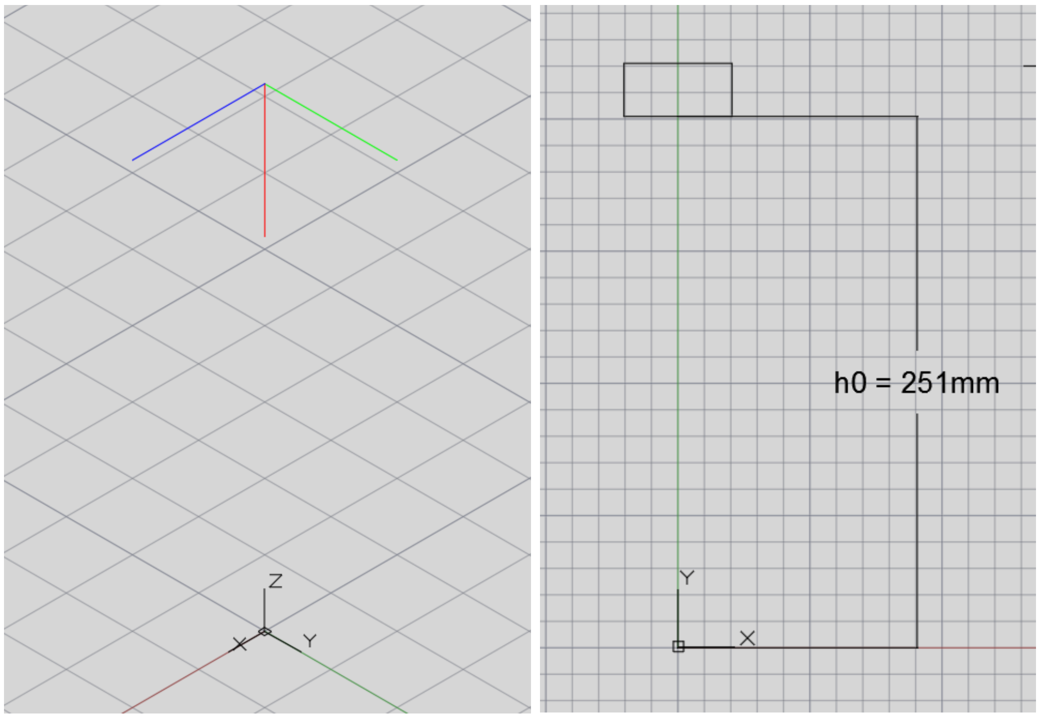

Figure 6 Scanner position at balance

When the roll angle changes, two scenarios could occur: the roll angle value is positive or negative when the car rolls to the right or to the left, respectively. When pitch changes, the rotation axis is identified as the line going through the rear wheel contacts with the ground. Taking the intersection of the rolling axes and the pitch rotating axis, two rotation center are identified, one at the right rear wheel contact with the ground and one at the left rear wheel contact with the ground. Figures 7 and 88 describe the axis of rotation of roll and pitch angles.

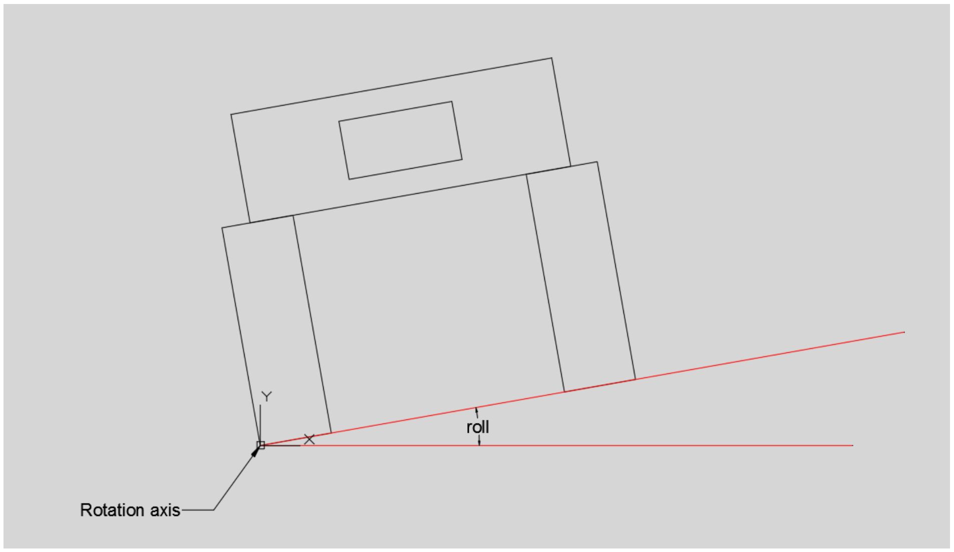

Figure 7 Roll angle rotation

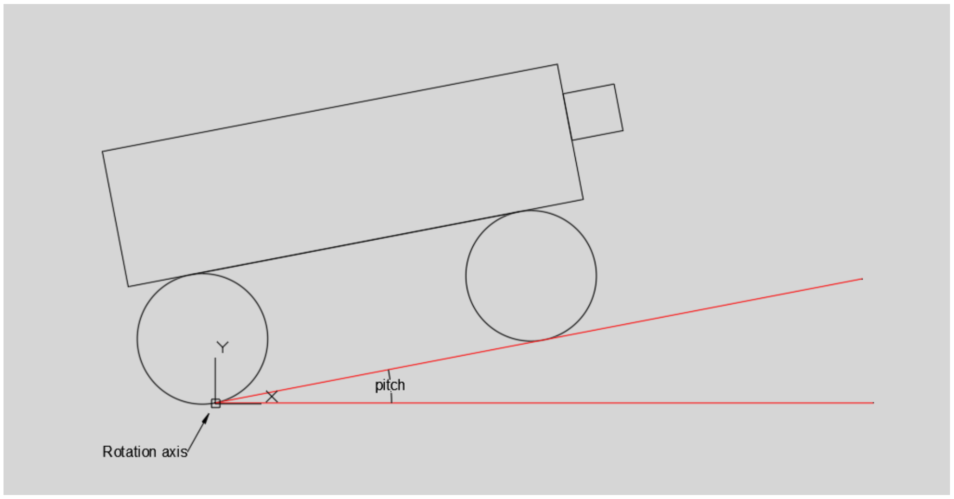

Figure 8 Pitch angle rotation

As an example, when both the roll angle and pitch angle are positive, the transformation operations are applied as following:

- First, assume that the scanner is at the origin of world coordinates
- Rotate the scanner around the X axis 90 degrees, in order to have the scanner orientation facing down to the ground
- Since the roll angle is positive, the right rear wheel is chosen as the rotation point. Therefore, the scanner is translated by a vector ***v*** which points to the vector from the wheel to the scanner. This vector is constant regardless of the orientation variation.
- Following this, the matrix which represents the rotation around the y-axis, with the roll as rotating angle is applied to the scanner. Then, the rotation matrix around the x-axis with pitch as rotating angle is applied.
- Next, the rotating point (the contact of the wheel with ground) is translated back to its original place by using vector ***-v***
- Finally, the scanner is moved to a fixed height h_0 by another translation operation, to represent its actual mounting position
- For each 2D scan, the car was different positions along its track. Therefore, the scanner position for each time a scan was taken must be displaced from the initial position by a certain amount along the moving track. This displacement equals to the velocity of the car times 1/15 second, which is the sampling period of the TiM561.

After these operations, the transformation of the scanner in world coordinates with correction of roll and pitch angles is applied. A matrix representing all these operations can be found by multiplying the individual transformation matrices in the reversed order.

Points transformation in world coordinates:

By multiplying the two matrices calculated above, the coordinates of each data point in the world coordinates is obtained. These values are used for visualization.

2.4.4. Visualization

Besides the coordinates vector, SGApp also used a vector to represent the colors of each point. The color scale ranges from red to blue, corresponding to the height of the measured points, from 60 to -60 millimeters. The lower the altitude, the bluer the color; the higher the altitude, the redder the color.

The velocity displacement can be adjusted in the menu Settings → Velocity. The data points vectors are then recalculated and rendered with the new displacement between each 2D scan.

Figure 9 shows an example of visualization of a record file.

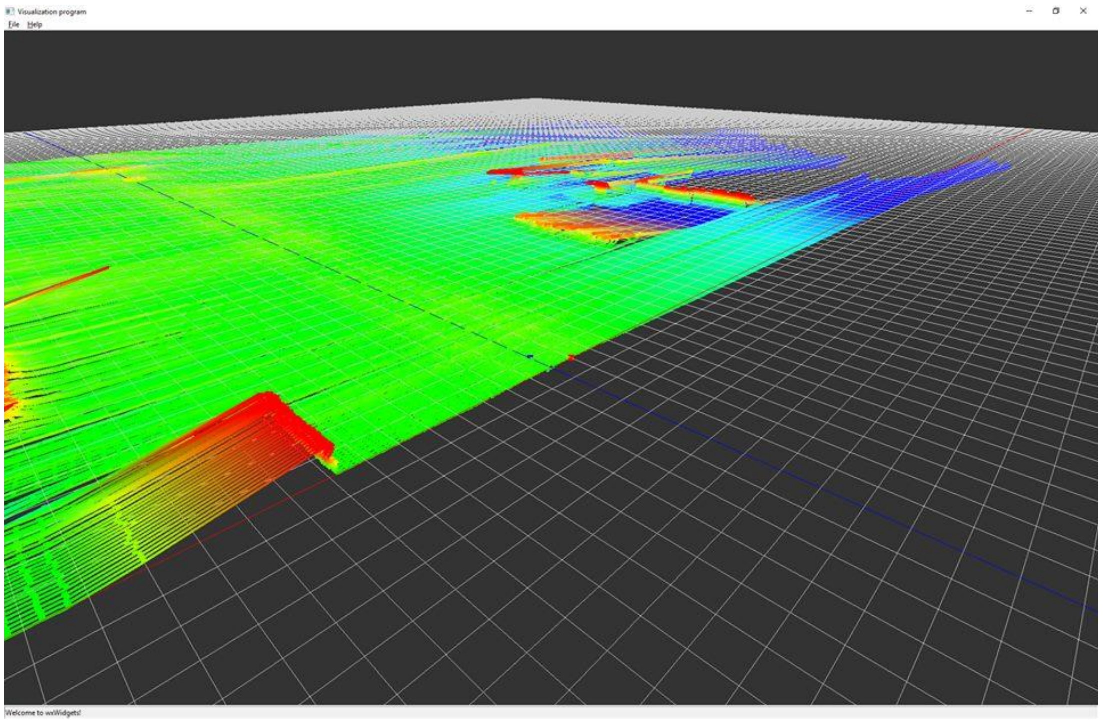

Figure 9 Visualization screen

3 Current limitations and improvement possibilities

The mechanical limitations of the vehicle were met quite hard in development. Solutions to most of them exist and could be implemented by changing or adding components. For this project, it was impossible to acquire the best possible components due to time constraints, but in the end the parts used worked adequately.

Mechanical aspects which could be improved, include:

- Using motors designed for wheel rotation with higher RPM for faster operation
- Commercially sold high resolution encoders, instead of manually fabricated ones for all wheels to enable four-wheel-drive and better speed measurements.
- A less tense, more advanced suspension system, air inflated wheels for reduction of mechanical vibration and improving energy efficiency
- Lighter LiPo (Lithium-Polymer) batteries.

The cost for these renovations would be insignificant compared to the benefits brought about. The performance boost attributed to these improvements include greatly improved controlling and steering abilities; weight reduction, longer operation time and less measurement noise due to vibration of the car body.

Currently, due to the quality of the encoders used, the measured velocity of the car is not usable in the visualization program. Through observation and external measurements, the car is running at a constant speed. The velocity value used in the visualization program was instead the setpoint value, as the observed velocity was not much different from the velocity setpoint.

With respect to the software side, the following expansions should be considered:

- A computer vision system, for more detailed environment data and object recognition applications.
- A user interface for remote control and steering, streaming live data feed from the car including the laser scans and the vision.
- GPS for detailed global position.

With the addition of these expansions, the car could be operated remotely or even automatically. For safety reasons, it is recommended to consider the operating environment carefully before implementing automatic control.

The addition of a vision system presents the possibility for machine-learning applications, where the image feed and the scan feed could be combined to derive the quality of the road surface.

4 Innovation - Conclusion

The ESAV works remarkably well even with a single measuring sensor, but if more resources were available, adding another TiM scanner would result in a measuring accuracy boost. As mentioned earlier, using one TiM scanner has limitations in the sparsity of the measurement point due to its angular aperture design, and having another (or more) equally powerful sensor would remove this limitation. For road surface scanning applications, this project demonstrated the capabilities of the laser scanner to replace the surface profiling measurement device.

Due to the weather conditions, outside testing could not be conducted. As the tests conducted were mostly indoors, a whole another application of the system created was also discovered in indoors room mapping. This application is promising, as the accuracy and performance of the TiM561 exceeded the teams' expectation. Accompanied by a vision system and a remote-control system with steering capabilities for navigation, highly complicated structures can be measured and reconstructed in the visualization program. The data collected can be used for multiple purposes such as maps for indoor robots; building verification and environment reconstruction in software applications. With the rise of data science and the Internet of Things, the possibilities for using this data is very promising and wide-open.

To conclude this report, the ESAV achieved its initial goals and opened up a world of possibilities for usage of the data acquired. The TiM561 achieved the accuracy of ± 15 mm, which was accurate enough for terrain mapping applications.

References

Abyarjoo F., Barreto A., Cofino J. & Ortega F. (2012). Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magenetic Sensors. CISSE 2012.

Finnish Transport Agency (2015). Finnish Road Statistics. Lappeenranta, Finland

Finnish Transport Agency. Finnish Road network. Retrieved November 29, 2016 from http://www.liikennevirasto.fi/web/en/road-network#.WD3rA_l95DA

Gautam, S (2014). Machine Design and Vision Based Navigation. HAMK University of Applied Sciences, Finland.

IMU Data Fusing: Complementary, Kalman, and Mahony Filter. Retrieved November 29, 2016, from http://www.olliw.eu/2013/imu-data-fusing/

Madgwick, S. O., Harrison, A. J., & Vaidyanathan, R. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm. 2011 IEEE International Conference on Rehabilitation Robotics. doi:10.1109/icorr.2011.5975346

Mahony, R., Hamel, T., & Pflimlin, J. (2005). Complementary filter design on the special orthogonal group SO(3). Proceedings of the 44th IEEE Conference on Decision and Control. doi:10.1109/cdc.2005.1582367

NASA (2015). Aircraft Rotations Body Axes. Retrieved November 29, 2016, from https://www.grc.nasa.gov/www/k-12/airplane/rotations.html

Appendices

The documentation, programs and source code documentation are attached in the submission, including:

- Car Control – Arduino program
- SGPiApp – Raspberry Pi program
- SGApp – Visualization program
- Brochures presenting the car and it’s functionality