Point-to-Point Navigation
for a Cost-Efficient Agricultural Robot

Axel Willekens, Geoffrey Ottoy*, Simon Cool†

Abstract – **To reduce the labour cost and operating time on the field, there is an ongoing trend towards heavier and wider machinery in the agriculture. This, however, has a negative impact on soil quality, structure and processes, which result in yield reduction [1]. Multiple small scale autonomous robots could solve these problems, while reducing the labour cost even more. This article focuses on the development of a point-to-point navigation system for a price-efficient robot, only using a Real-Time Kinematic Global Positioning System (RTK-GPS) and an Inertial Measurement Unit (IMU) as sensors. The implementation of the algorithm will be done in Robot Operating System (ROS)[1], from which the core is running on a Raspberry Pi. A Fuzzy Logic [2] and Sliding Mode Control (SMC) [3] algorithm have been evaluated in simulations. The SMC-based control algorithm benefits an easy implementation and depends on very little parameters. The Fuzzy Logic algorithm is more complex to implement, but is more intuitive for the programmer. Because of the good stability and the limited set of implementation parameters the SMC-based algorithm is chosen to implement on the robot platform. This is tested under varying conditions (in the field).**

Index Terms – **Agricultural robot, Autonomous point-to-point navigation, sliding Mode Control, Fuzzy logic, Lyapunov functions**

I Introduction

Precision Agriculture aims to increase the efficiency in the agriculture by employing technology on the field and in greenhouses. That is where robotics came up. Robots are able to execute tasks precisely and accurate. While cultivation treatments are done, robots can gather information of the soil, the crops and the environment which can be used to make better decisions. Where the classical agriculture focuses on the field its needs, precision agriculture looks at the needs of each individual crop and the local soil condition. This results in a more efficient production with less environmental impact.

Next to the precision agricultural aspect robotics could solve the problem of soil compaction. To reduce the labour cost and operating time on the field, there is an ongoing trend towards heavier and wider machinery in the agriculture. This, however, has a negative impact on soil quality, structure and processes, which result in yield reduction and lowers the biodiversity. Multiple small scale autonomous robots could solve these problems, while reducing the labour cost even more.

Considering these two reasons robotics started to make their entrance in the agriculture in the recent years. Hereby autonomous navigation seems to be a difficult and often underestimated challenge. To navigate a robot autonomously in a greenhouse, an orchard or a crop field different approaches can be made. A first example makes use of visual row detection whereby the robot drives in between the crop rows using camera images. A technique in development at the Flemish Institute of agriculture and fishing research (ILVO). Another example investigated at the university of Wageningen uses Laser Imaging Detection And Ranging (LIDAR) to navigate a robot in between the tree rows of an orchard. To make the introduction of robotics possible at a large scale in the agriculture, it is crucial that accurate navigation methods are developed without the need of plant- or tree rows. An easy to access infrastructure to achieve this is Global Positioning System (GPS). An example of a robot that realizes autonomous navigation with GPS is the Smart Weeding Robot from Ecorobotix. Similar projects in the academic world describe that the minimum required hardware for autonomous navigation is a GPS and Internal Measurement Unit (IMU) [4][5]. These can detect the robot position, the orientation and the linear and angular velocity of the robot. This is briefly called the state or posture of the robot and can be expressed in a matrix $P = (X, v_X, Y, v_Y, \theta, \omega)^T$.

To find an appropriate algorithm for autonomous point-to-point navigation on a farming field, two algorithms will be extensively investigated, simulated and tested on their stability. Autonomous control algorithms implements mathematical methods, such as Proportional Integrated Differential (PID) - tuning or Lyapunov functions, to detect if the system goes to its equilibrium. Both mathematical mechanisms defines certain stability criteria which are able to detect whether the system is in a stable state or not.

The first algorithm, investigated and simulated in this paper, uses Lyapunov functions to check the stability of the system. Sliding Mode Control (SMC) pushes the robot in the stable state and makes sure the asymptotically stability theorem of Lyapunov is met.

The second algorithm uses the PID theorem as mathematical method to check the stability of the system. Fuzzy logic is used to alter the PID - parameters depending on the current state of the robot. It is not possible to use a fixed set of PID - parameters because of the non - linearity of the system [6].

After the *Preliminary study* where the SMC based and Fuzzy PID based algorithm are investigated and simulated in Simulink (Matlab). The best algorithm is picked to implement on the test robot platform. This will be discussed in the section *Implementation*. The software is implemented in Robot Operating System (ROS) and will run on a Raspberry Pi (RPI). Next to the RPI the robot platform contains a Real-Time Kinematic Global Positioning System (RTK-GPS) and an Inertial Measurement Unit (IMU) as sensors and four brushless DC motors each driven by its motor driver. The GPIO's of the RPI together with a Digital - to - Analog (DAC) are used to control de motor drivers. Additional to the control algorithms Kalman filtering will be used to increase the update

*KU Leuven Technologiecluster Elektrotechniek (ESAT)

†Instituut voor Landbouw-, Visserij- en Voedingsonderzoek (ILVO) eenheid technologie en voeding

[1]ROS website: `http://www.ros.org/`

frequency. In *Results and Discussion* the algorithm will be tested on the field. Afterwards a *Conclusion* will be drawn.

II Preliminary Study

II.1 Sliding Mode Control (SMC)

SMC as trajectory tracking control algorithm has a fast response and can handle uncertainties from the system itself due to disturbances on the sensor data. SMC is a not linear control strategy where, in this case, a switching function is composed starting from a stable Lyapunov function. The switching function has to converge to zero, only then the system is in sliding mode and is Lyapunov stable.

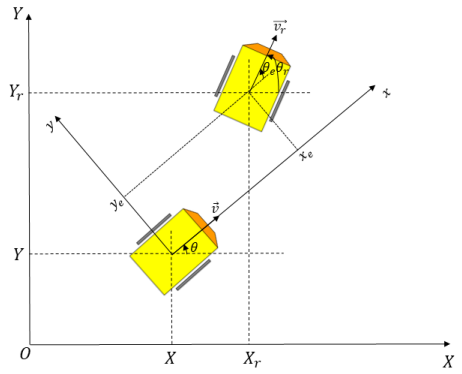


Figure 1: Coordinate-system robot using SMC

The error matrix of the robot is defined as in figure 1:

$$p_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_r - X \\ Y_r - Y \\ \theta_r - \theta \end{bmatrix} = \begin{bmatrix} cos(\theta)\cdot(X_r - X) + sin(\theta)\cdot(Y_r - Y) \\ -sin(\theta)\cdot(X_r - X) + cos(\theta)\cdot(Y_r - Y) \\ \theta_r - \theta \end{bmatrix} \quad (1)$$

Derivation of equation 1 to the time gives rise to the speed error matrix:

$$\dot{p_e} = \begin{bmatrix} \dot{x_e} \\ \dot{y_e} \\ \dot{\theta_e} \end{bmatrix} = \begin{bmatrix} y_e\,\omega - v + v_r\,cos(\theta_e) \\ -x_e\,\omega + v_r\,sin(\theta_e) \\ \omega_r - \omega \end{bmatrix} \quad (2)$$

To create the sliding mode controller two steps have to be made.

1. The correct switching functions $s(t)$ has to be found. In this situation the sliding mode surface has to contain two switching functions $s_1(t)$ and $s_2(t)$ in order to receive two control laws, one for the linear speed v and one for the angular speed ω.

2. In a next step the control laws have to be constructed in a way that the system achieves sliding mode and remains in it ($s_1(t) = 0$ and $s_2(t) = 0$).

The process of finding the switching surface can be simplified by choosing the first switching function $x_e = 0$ [7]. The second switching function is constructed using Lyapunov functions. A Lyapunov candidate function $V(X) = V(x_e, y_e) = a \cdot x_e^2 + b \cdot y_e^2$ is chosen. Because the first switching function $x_e = 0$, this can be reduced to $V(y_e) = \frac{1}{2} \cdot y_e^2$ choosing $b = \frac{1}{2}$. Derivation to the time and substitution of equation 2 gives:

$$\dot{V_y} = y_e \dot{y_e} = y_e(-x_e\omega + v_r sin(\theta_e)) = -x_e y_e \omega - v_r y_e sin(\theta_e) \quad (3)$$

In [8] is proven that the third condition for a Lyapunov function ($\frac{dV}{dt} \leq 0 \;\; \forall \;\; X \in U$) is fulfilled if $\theta_e = -arctan(v_r y_e)$. Now Lyapunov stated that if in a neighborhood U of the zero solution $X = 0$ of an autonomous system there is a Lyapunov function $V(X)$ with a negative definite derivative $\frac{dV}{dt} < 0$ for all $X \in U \setminus \{0\}$, then the equilibrium point $X = 0$ of the system is asymptotically stable [9]. Considering this theorem, the system is Lyapunov stable and in the equilibrium $V_y \equiv 0 \Rightarrow x_e = 0$, $y_e = 0$ and $\theta_e = -arctan(v_r y_e) = 0$. This equilibrium will be achieved if $x_e \to 0$ and $\theta_e \to -arctan(v_r y_e)$. The switching surface can be written as:

$$s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} x_e \\ \theta_e + arctan(v_r y_e) \end{bmatrix} \quad (4)$$

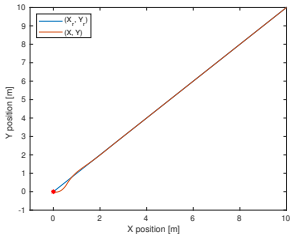
Whereas the switching surface is constructed, the appropriate control laws can be searched. According to the position of the system on the switching surface, the functions f_1 or f_2 will be executed. f_1 and f_2 are determined by the switching function $\dot{s} = \frac{ds}{dt}$. Consider $\dot{s} = -k \cdot sat(s)$, where $sat(s)$ is a saturation function. Then this will have to match with the derivation of the switching mode functions described in equation 4.

$$\begin{bmatrix} \dot{s_1} \\ \dot{s_2} \end{bmatrix} = \begin{bmatrix} -k_1 \cdot sat(s_1) \\ -k_2 \cdot sat(s_2) \end{bmatrix} = \begin{bmatrix} \dot{x_e} \\ \dot{\theta_e} + \frac{y_e}{1+(v_r y_e)^2}\dot{v_r} + \frac{v_r}{1+(v_r y_e)^2}\dot{y_e} \end{bmatrix}$$
$$\Leftrightarrow \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} y_e\,\omega + v_r\,cos(\theta_e) + k_1 \cdot sat(s_1) \\ \omega_r + \frac{y_e \cdot \dot{v_r}}{1+(v_r y_e)^2} + \frac{v_r^2 \cdot sin(\theta_e)}{1+(v_r y_e)^2} + k_2 \cdot sat(s_2) \end{bmatrix} \quad (5)$$

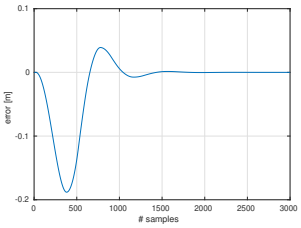
Equation 5 describes the control laws only having two parameters k_1 and k_2. Implementation of these control laws in a Simulink model gives rise to an error as shown in figure 2b when the system travels along a trajectory shown in figure 2a.

Given the simulation results, the SMC algorithm can be considered as stable. Also when artificial noise is added in the Simulink scheme the error remains low, which proves the robustness of this algorithm [10].

In the above simulation the required angular speed ω_r is proportional to θ_e using a third parameter k_3. In this way the responsivity of the system is increased even more reducing the overall error.



(a) Trajectory robot with the red star the starting position



(b) Error in respect to ideal trajectory [m], positive left – negative right from trajectory

Figure 2: SMC – starting position $P = (0,0,-\frac{\pi}{4})^T$ using $k_3 = 1$

II.2 Fuzzy PID

A second algorithm that was intensively researched is one based on Fuzzy logic. Where most of all binary values are considered (true or false, 1 or 0), Fuzzy logic uses Membership Functions (MF) in an interval $[0,1]$ [11]. Here a fuzzy logic network is developed where the fuzzy sets depends on θ_e and d_e, as defined in figure 3. The linear velocity v and the variation on the PID - parameters to determine the angular velocity ω are calculated using input set MFs for θ_e and d_e. Triangular type-1 Fuzzy set MFs are used, these should satisfy the real-time requirements [12]. Centroid deffuzication is used to determine the outcome. The fuzzy rules are composed using logic reasoning. For example when $\theta_e \approx 0$ and d_e is not, the robot is driving with an offset error along its path and it is appropriate to enlarge the K_i parameter. For the linear velocity v, the robot should slow down when θ_e is large. The lower the speed, the lower the eventual error in respect to the ideal trajectory.

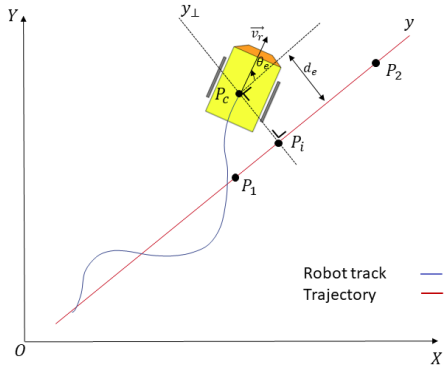
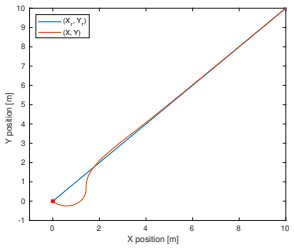
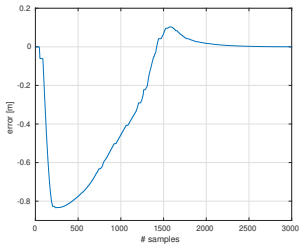


Figure 3: Defenition θ_e and d_e in Fuzzy logic algorithm

In simulation the stability of the algorithm is proven. The error in respect to the ideal trajectory (figure 4b) is smaller compared to a same constellation using PID - tuning for the angular velocity and a constant speed for the linear velocity as control laws [10]. In comparison with the SMC algorithm, the fuzzy algorithm is less responsive. Another big disadvantage is the amount af parameters that has to be set. The input MFs for θ_e and d_e, the output MFs for v, dK_p, dK_i, dK_d and the fuzzy rules has to be carefully chosen. This makes the implementation of this algorithm on a robot platform very inefficient.



(a) Trajectory robot with the red star the starting position



(b) Error in respect to ideal trajectory [m], positive left – negative right from trajectory

Figure 4: Fuzzy PID – starting position $P = (0,0,-\frac{\pi}{4})^T$

II.3 Kalman filtering

Using Kalman filtering the update frequency can be increased, above all the measured values with a large variance are middled out. The Kalman estimation is done on a timestamp $t = k \cdot T_s$, for simplicity we write $t = k$.

On timestamp $t = k$

- The robot state $x_k = (X_k, v_{X,k}, Y_k, v_{Y,k}, \theta_k, \omega)^T$
- The observation $z_k = (X_{GPS,k}, v_{X,GPS,k}, Y_{GPS,k}, v_{Y,GPS,k}, \theta_{IMU,k}, \omega_{IMU,k})^T$
- The action $u_k = (v_{k,action}, \omega_{k,action})^T$

Instead of doing $x_k = z_k$ on timestamp $t = k$ a Kalman estimation is made. A Kalman model is used to calculate the next state, consequently a gain K_k is calculated which determines the weights of the calculated state and the measurements.

The kalman model used in this study is given in equations (6) to (9).

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{6}$$

$$\Leftrightarrow \begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \\ \theta_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T_s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{k-1} \\ v_{X,k-1} \\ Y_{k-1} \\ v_{Y,k-1} \\ \theta_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} cos(\theta)\,T_s & 0 \\ cos(\theta) & 0 \\ sin(\theta)\,T_s & 0 \\ sin(\theta) & 0 \\ 0 & T_s \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k,actie} \\ \omega_{k,actie} \end{bmatrix} + w_k \tag{7}$$

$$z_k = H_k x_k + v_k \tag{8}$$

$$\Leftrightarrow \begin{bmatrix} X_{GPS,k} \\ v_{X,GPS,k} \\ Y_{GPS,k} \\ v_{Y,GPS,k} \\ \theta_{IMU,k} \\ \omega_{GPS,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \\ \theta_k \\ \omega_k \end{bmatrix} + v_k \tag{9}$$

This model gives rise to good simulation results. In figure 5 and figure 6, the results of using Kalman filtering in a Simulink simulation is shown for the X position and the Y position respectively. In the first graph the Kalman estimation and the exact value are plotted. The second graph plots the observation noise and the error due to Kalman estimation.

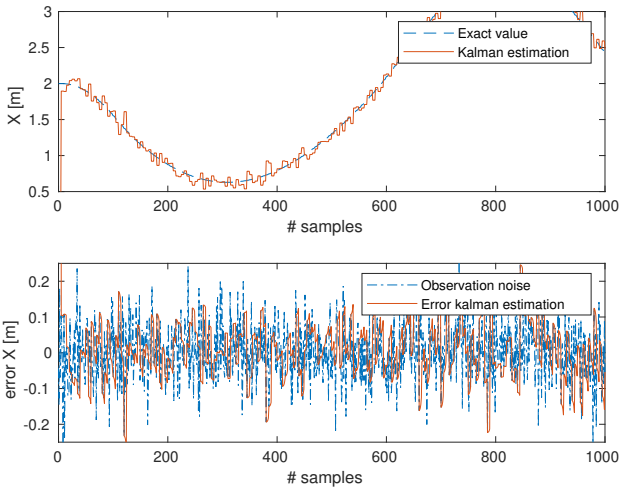


Figure 5: Kalman filter response for X

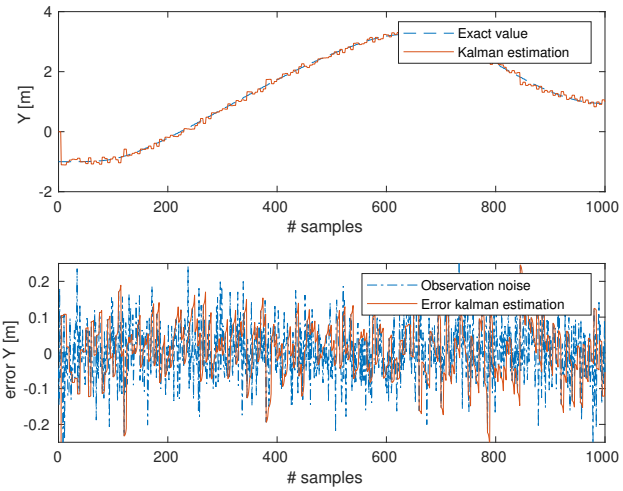


Figure 6: Kalman filter response for Y

The superposed observation noise in simulation is Gaussian and is added to the X, Y, v_X, v_Y, θ and ω value. When the variance of these observation noises gets to small, the error of the Kalman estimation will overrule the observation noise in the lower graph of figure 5 and figure 6. By then the raw sensor data are more accurate than the Kalman estimation, consequently Kalman filtering is not usefull anymore.

III Implementation

The SMC based algorithm will be implemented. According to the simulation, it should be stable and in comparison with the Fuzzy PID algorithm the amount of tunable parameters is rather small. Above all the total error (figure 2b) remains smaller than the error of the Fuzzy PID algorithm (figure 4b). This algorithm will be combined with Kalman filtering to increase the update rate and smooth out inaccurate measured values. The software implementation will be done in ROS, a robot software development platform in which the programmer is able to abstract hardware- and software functions into nodes. ROS makes communication between these nodes possible. This increases the modularity of a robot program. Above all the nodes can be tested individually.

III.1 Hardware

An illustration of the robot can be seen in figure 7. As mentioned in the introduction, the four wheels are driven by brushless DC motors (type EC-max 40 serie from Maxon motor) each driven by its motor driver (type 4-Q-EC EC DECV 50/5 305259 from Maxon motor). The RTK - GPS is centrally taped on the robot. Considering the speed measurements done with it, the orientation is important. The antenna of the Emlid Reach is mounted on the electricity box at the upper left side of the robot. This electricity box contains the Raspberry Pi and an LTC1661 DAC. The Razor 9-Degrees of Freedom (DoF) IMU from Sparkfun is mounted at a certain level above the platform to reduce iron interference after calibration. The 48 V battery is mounted under the platform. The Raspberry Pi is powered with a power bank which can be found at the upper right corner of the robot platform. On the power bank a device, to connect with the 4G network, is mounted. From this device the RTK - GPS receives its Flemish Positioning Service (FLEPOS) - corrections. This hotspot can also be used to *ssh* on the Raspberry Pi for debugging purposes.

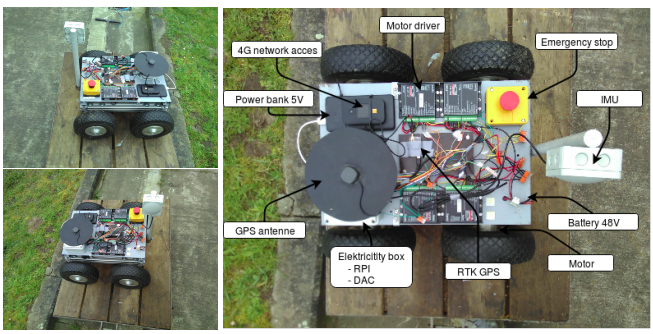


Figure 7: Robot hardware implementatie

III.2 Software

An overview[2] of the ROS nodes and topics is given in figure 8. The two services, *Angle.srv* from the *odometry_source* to the *imu_driver* and *GetPoint.srv* from the *odometry_source* to the *point_creator* are indicated with dotted arrows.

[2]ROS catkin_ws and Simulink simulation files can be found on
`https://github.com/axelwillekens/Point-to-Point-Navigation.git`

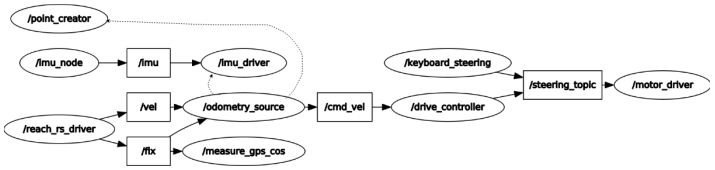


Figure 8: Rosnodes (ovals) and rostopics (rectangles, solid arrows)

The *reach_rs_driver* [13] publishes *sensor_msgs/NavSatFix* on the $\sim$*fix* topic and *sensor_msgs/TwistStamped* on the $\sim$*vel* topic. These messages are composed using the National Marine Electronics Association (NMEA) sentences from the GPS. The 3D location and its covariance matrix are extracted from the Global Positioning System Fix Data (GPGGA) sentence and added in a *sensor_msgs/NavSatFix*. The Global positioning Recommended minimum specific GPS/Transit data (GPRMC) sentence is used to get information about the speed to compose a *sensor_msgs/TwistStamped*.

The *imu_node* [14] is necessary to connect to the 9DoF Razor IMU M0 on the Sparkfun breakout board. On the topic $\sim$*imu*, a *sensor_msgs/Imu* is published which contains a quaternion (to indicate the orientation of the robot), the angular and the linear velocity. To ensure a good operation of the IMU, a callibration has to be done. The *imu_driver* supports the ROS service *Angle.srv*. In this method, the quaternion is mapped to euler angles, which is necessary because the yaw is used in the algorithm. To get the yaw, the consequent formula is used [15]:

$$\theta = arctan2(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2))$$

The *drive_controller* listens to the *cmd_vel* topic on which the *odometry_source* publishes *geometry_msgs/Twist* which contains the linear and angular speed action the robot should take. These values are mapped to v_l and v_r, respectively the speed of the left and the speed af the right wheels $\left[\frac{m}{s}\right]$. Using figure 9 the following equations for v_l and v_r can be found:

$$v_l = \frac{2 \cdot v_x}{\frac{R+a}{R-a} + 1} \text{ met } R = \frac{v_x}{\omega_z}$$

$$v_r = \frac{2 \cdot v_x}{\frac{R-a}{R+a} + 1} \text{ met } R = \frac{v_x}{\omega_z}$$

waarbij $\omega_z \circlearrowleft > 0$ en $\omega_z \circlearrowright < 0$

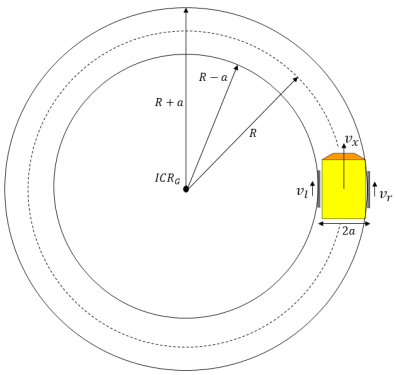


Figure 9: Illustration to determine v_l and v_r

The values for v_l and v_r has to be mapped to a Digital-to-Analog Converter (DAC) value. The DAC used in this project is the LTC1661. With a tachometer the relationship between the speed $\left[\frac{m}{s}\right]$ and the DAC value $[/]$ can be found. With Logger Pro, the following linear fit is calculated:

$$DAC\,value = 421.3 \cdot v - 71.43$$

These DAC values are published on the *steering_topic*. The *motor_driver* listens to this topic and put these values on the GPIOs of the RPI using a *wiringPi* and *LTC1661* library [16][17].

The node *point_creator* updates the points on which the robot drives (trajectory points). To calculate the next point, the current position and orientation are passed as parameters for the *GetPoint.srv*. x_e, y_e and θ_e are then returned to the *odometry_source*.

The *odometry_source* implements the control laws from the SMC algorithm (equation 5). Using Kalman filtering the update frequency can be increased form 10 Hz to 20 Hz.

The other two nodes: *measure_gps_cos* and *keyboard_steering* are needed for debugging purposes.

III.3 Evaluation

Two different approaches are used to evaluate the robot implementation of the SMC algorithm on the field. For a linear track, the equation of the line is used to calculate the shortest distance between each measured point of the robot trajectory and the track. For non-linear tracks the error is also defined as the shortest distance between a measured point and the track. Here the error is found in an iterative way.

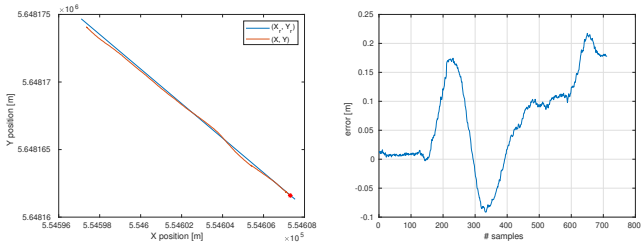
IV Results and discussion

The error of the implementation of the SMC algorithm depends on the following parameters:

- k_1 and k_2: the parameters in the SMC control laws.
- $d_{traject}$: the distance between the trajectory points (the points where the robot drives to).
- d_{switch}: the moment the robot is d_{switch} $[m]$ removed from a point, the trajectory point is updated.
- update frequency: can be increased using Kalman filtering.
- required linear speed v_r and angular speed ω_r. ω_r varies proportionally with θ_e using k_3.

The result of a linear trajectory for a certain parameter set is given in figure 10. The best founded parameter set gives rise to an absolute error of ≈ 20 cm.

To evaluate the responstime of the system, non-linear trajectories are followed. To achieve good responsivity a different parameter set has to be chosen. At a non-linear trajectory the error remains lower than 50 cm. The results can be seen in figure 11 and figure 12.

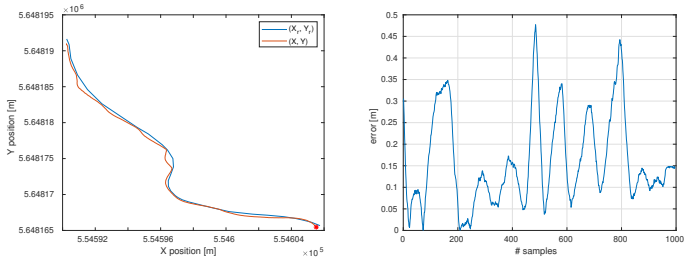


(a) Trajectory robot in UTM coordinate reference with the red star the starting position

(b) Error in respect to ideal trajectory [m], positive left – negative right from trajectory

Figure 10: Accuracy test

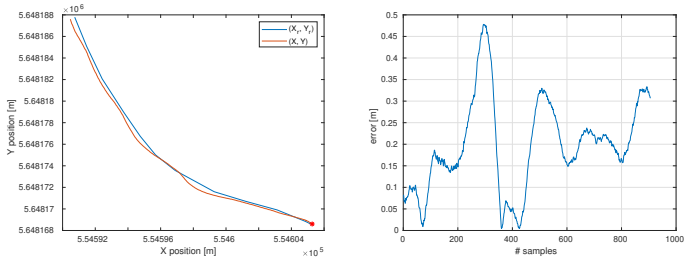
The accuracy of the robot implementation on a linear track is sufficient to navigate between crop rows which are planted 50 to 75 cm from each other. Yet a higher accuracy is desirable for some cultivation operations e.g. weeding (in the row). For these operations a sensor fusion will be needed, combining GPS, IMU and visual images for more precise navigation.



(a) Trajectory robot in UTM coordinate reference with the red star the starting position

(b) Error in respect to ideal trajectory [m]

Figure 11: Responsivity test – trajectory 1



(a) Trajectory robot in UTM coordinate reference with the red star the starting position

(b) Error in respect to ideal trajectory [m]

Figure 12: Responsivity test – trajectory 2

Figure 11 and figure 12 suggest that the algorithm is responsive enough to handle unexpected conditions in the field. Yet not responsive enough to handle sharp 90° corners. To turn on the headland a different approach will have to be investigated.

V Conclusion

In this study different algorithms for autonomous point-to-point navigation are investigated only using a GPS and an IMU. The SMC trajectory algorithm and the fuzzy PID trajectory algorithm are simulated in Simulink. Both algorithms seemed to be stable but the SMC algorithm gave the best results. Also the limited amount of tunable parameters seems very interesting for implementation.

For a good chosen parameter set an accuracy of 20 cm is achieved on linear tracks. Accurate enough to navigate between crop rows which are planted 50 to 75 cm from each other. Yet a higher accuracy is desirable for some cultivation operations e.g. weeding (in the row). For these operations a sensor fusion will be needed, combining GPS, IMU and visual images for more precise navigation.

For non-linear tracks a higher responsivity is configured at the expense of the accuracy, but still the maximum error remains under 50 cm. The responsivity is not large enough to conquer 90° corners. A different approach to turn on the headland has to be investigated.

We can conclude that a maximum absolute error of 20 cm is accurate enough to navigate between crop rows, which has mostly an interrow distance of 50 to 75 cm, though a higher accuracy is desirable.

The SMC algorithm can be used to follow linear lines on the field. To turn on the headland another turning algorithm should be developed. Further research can check if other Lyapunov functions can increase the accuracy. Accuracy may also be gained by improving the kinematic model.

REFERENCES

[1] E. Lie E. Fugelsnes. The research council of norway stensberggata. Stensberggata 26, Oslo, 2011. The Research Council of Norway.

[2] C. C. Lee. Fuzzy logic in control systems: fuzzy logic controller. ii. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):419–435, March 1990.

[3] Jingxin Shi Vadim Utkin, Juergen Guldner. *Sliding Mode Control in Electro-Mechanical Systems*, volume 2. Boca Raton: CRC Press, 2009. Geometry of manifolds with special holonomy: '100 years of holonomy'.

[4] Liangliang Yang Chi Zhang, Noboru Noguchi. Leader-follower system using two robot tractors to improve work efficiency. *Computers and Electronics in Agriculture*, (121):269–281, 2016.

[5] Claude Lague Chengming Luo, Ahmad Mohsenimanesh. Parallel point-to-point tracking for agricultural wide-span implement carrier (wsic). *Computers and Electronics in Agriculture*, (153):302–312, 2018.

[6] Hoon Lim Jang Myung Lee Jun Ho Lee, Cong Lin. Sliding mode control for trajectory tracking of mobile robot in rfid sensor space. *International Journal of Control, Automation, and Systems*, 7(3):429–435, 2009.

[7] M. Salehi and G. Vossoughi. Impedance control of flexible base mobile manipulator using singular perturbation method and sliding mode control law. *International Journal of Control, Automation, and Systems*, 6(3):677–688, 2008.

[8] E. Kayacan, E. Kayacan, H. Ramon, O. Kaynak, and W. Saeys. Towards agrobots: Trajectory control of an autonomous tractor using type-2 fuzzy logic controllers. *IEEE/ASME Transactions on Mechatronics*, 20(1):287–298, Feb 2015.

[9] math24. Method of lyapunov functions. `https://www.math24.net/method-lyapunov-functions/`, 2018. consulted the 21/10/2018.

[10] A. Willekens S. Cool, G. Ottoy. Point-to-point navigatie voor een kostenefficiënte robot in de landbouw. `https://github.com/axelwillekens/Point-to-Point-Navigation`, May 14, 2019. consulted the 14/05/2019.

[11] Mojtaba Ahmadieh Khanesar Erdal Kayacan. *Fuzzy Neural Networks For Real Time Control Applications*. Joe Hayton, 2016.

[12] Shanan Chen Shengqi Yan Qing Xu, Jiangmin Kan. Fuzzy pid based trajectory tracking control of mobile robot and its simulation in simulink. *International Journal of Control and Automation*, 7(8):233–244, 2014.

[13] enwaytech. reach_rs_ros_driver. `https://github.com/enwaytech/reach_rs_ros_driver`, Jun 29, 2018. consulted the 03/03/2019.

[14] KristofRobot. razor_imu_9dof. `https://github.com/KristofRobot/razor_imu_9dof`, Jan 20, 2018. consulted the 03/03/2019.

[15] NASA. Euler angles, quaternions and transformation matrices. `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770024290.pdf`, 1997. consulted the 13/04/2019.

[16] Gordon Henderson. Wiring pi. `http://wiringpi.com/`, 2018. consulted the 10/03/2019.

[17] Stefan Wallnoefer. Ltc1661. `https://github.com/walle86/LTC1661`, 2014. consulted the 10/03/2019.