The Autonomous Flying Vacuum

1st Jan-Ruben Schmid
Det Tekniske Fakultet
University of Southern Denmark
Odense, Denmark
jschm20@student.sdu.dk

2nd Nathan Durocher
Det Tekniske Fakultet
University of Southern Denmark
Odense, Denmark
nadur20@student.sdu.dk

***Abstract*—In this paper we investigate the problem of implementing a real-time image-based feedforward controller on a quadcopter. A method was developed with a focus on extracting a desired trajectory from the environment using an on-board camera and applying a feedforward control method to navigate a course. The image processing and controls were designed around having a distinct line to follow and a simplified dynamical model. Through extensive simulation development our method showed promising results, while the deployment to hardware was unsuccessful due to hardware limitations.**

***Index Terms*—Feedforward control, Hough transform, MATLAB, Autonomous drone**

I. Introduction

DroneII, a leading drone market intelligence company, forecasts the drone industry revenue in the commercial sector will grow by an average rate of 13.8 percent until 2025 [1]. The expected growth rate in the beyond visual line-of-sight (BVLOS) drone market is even higher at 15 percent regarding a report form 2020, published by Business Wire [2]. The European Union published a master plan which targets autonomous flights, where the initial service will be available in 2022, the full service is scheduled to be operational in 2035 [3, p33]. Therefore, the BVLOS flight will become routine with some constraints with the implementation of U-Space level 2, which is the second level of drone autonomy in the EU [3, p11]. This leads to an increasing amount of skilled people who need to understand the technology and its challenges. In light of this market and labour potential, we pursed a project that aims to be used in the BVLOS industry using intelligent behaviour and autonomous navigation.

The state of the art method for implementing autonomous control is feedforward control with many papers and articles describing robust implementations of aggressive maneuver control. Mellinger et al. [4] demonstrate extremely fast and accurate maneuvers, by defining multi-phase trajectories where the control point changes along the path. These phases defines a movement from a hover to a dramatic incline at high speeds and back to a hover. This method enables agile movement through narrow opening and precision landings on steep or inverted surfaces with a standard quadcopter. Similarly, Kumar et al. [5] detail a feedforward control method with linearization techniques for a tilt rotor quadcopter. By coupling the tilt of diagonally opposite rotors the dynamics simplify to six control inputs for six degrees of freedom. Using this they are able to achieve even tighter control than that of standard quadrotor with feedforward control.

In contrast to [4], [5], our approach does not rely on predefined trajectories with known global coordinates. This allows for a more practical and flexible solution as the quadcopter is able to determine in real-time the correct direction in which to proceed. Also, we simplified the dynamics of our drone to that of a differential drive mobile robot restricting the control inputs. This eliminated unnecessary control complexity given the desired line following task. The method was developed in a simulation environment, where a quadcopter attempts to follow a desired track while maintaining a constant height.

The report starts with a summary of the theoretical knowledge, which is applied throughout the implementation, like the control model of the drone, the differential flatness controller, as well as the Hough transform. Afterwards an overview is given of the implementation with a brief description of the state machine, where each part is described in detail. The performance is then evaluated in simulation and the real world. The results are discussed and concluded upon.

II. Control Model

A. Dynamics

The dynamical model we used was based on a nonholonomic differential drive car. Using the standard quadcopter body reference seen in figure 1, we constrain the model by fixing both the pitch (θ) and roll (ϕ) axes to zero. Similarly the altitude or Z component of the quadcopter is controlled only through a state machine (explained in section III-A) where it is set to take off, remain at a specific height and land.

This forces the quadcopter to rotate around the yaw axis, ψ, to perform turns. To further simplify the model we also restricted positional move to the body frame's x direction. This resulted in the dynamics seen in equation 1 where the X and Y coordinates are the environment position and Θ is the velocity vector direction in the X-Y plane. This is where an approximation is made as there will be a delay in changing of the velocity direction after the vehicle has pivoted to a new

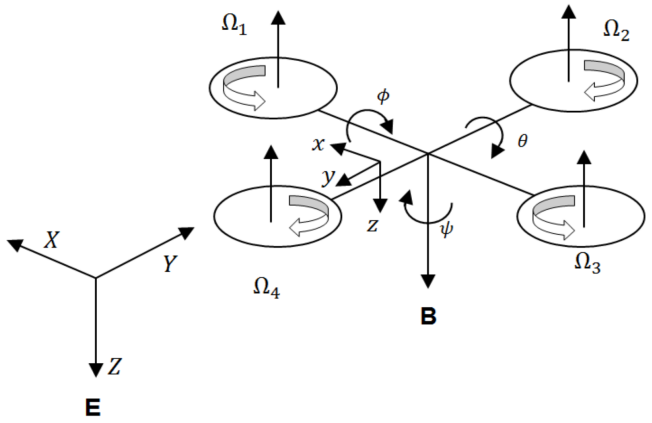

Fig. 1. A common definition of a dynamical model for a quadcopter[6], showing both the body frame and the environment frame.

direction when rotated along the yaw axis. However, at slower speeds we can assume this delay to be negligible. The input u_1 is the velocity in the body frame's x direction and the second input u_2 is the yaw angular velocity $\dot{\psi}$.

$$\dot{x} = \begin{cases} \dot{X} = u_1 cos(\Theta) \\ \dot{Y} = u_1 sin(\Theta) \\ \dot{\Theta} = u_2 \end{cases} \tag{1}$$

B. Flatness

Using this model, a system of differential flatness equations can derived by using the X and Y positions as flat variables.

$$F_1 = X \quad F_2 = Y \tag{2}$$

From here, we can write both the remaining state variable and the inputs in term of the flat variables.

$$x = \begin{cases} F_1 \\ F_2 \\ \arctan(\frac{\dot{F}_1}{\dot{F}_2}) \end{cases} \qquad u = \begin{cases} \sqrt{\dot{F}_1 + \dot{F}_2} \\ \frac{\ddot{F}_2 F_1 - \ddot{F}_1 F_2}{\dot{F}_1^2 + \dot{F}_2^2} \end{cases} \tag{3}$$

Where $\ddot{F}_i$ and $\dot{F}_i$ are the first and second time derivatives of the flat variables. Having both our state and inputs in terms of flat variables and their derivatives we then need to define them such that they are continuous and twice differentiable.

Using a center line of the desired track, extracted from our image processing explained in detail in section III, we can generate a third order polynomial trajectory that is parameterized by time for both F_1 and F_2.

C. Velocity Vs. Position Control

The Parrot minidrone control system package from MathWorks included a controller for the translational movement of the drone. The system can be configured in two different ways, either with position control or velocity control. Position control defines a global origin at the take-off location and then uses PID with the estimated XYZ position to pitch and roll, depending on the yaw orientation to reach the current set position, therefore the yaw orientation is not needed as a control input. Conversely, velocity control also uses a PID control to reach a given reference value but strictly effects the pitch and roll movements to move the drone. Here the yaw isn't necessarily required either since the drone can have independent X and Y velocities. Our decision to use a velocity control was due to the fact that the position would have to be constantly tracked and our approach was intended to less dependent on global environment knowledge.

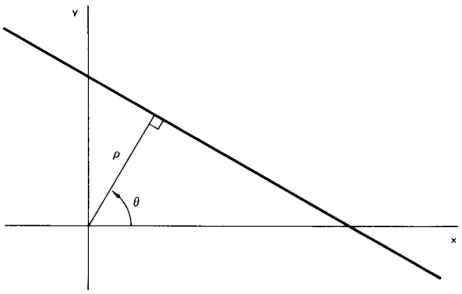

Fig. 2. Normal parameters of straight line [8]

D. Hough transform

The Hough transforms is a method to find circles and straight lines within an image. As input, a black and white image that had been processed to only show edges is required. This is done by a common edge detection algorithms, such as Sobel, Canny, Prewitt, Roberts or a fuzzy logic method [7]. The Hough transform computes a line by, creating a straight line for all θ values, shown in equation 4 at every edge pixel. The remaining edge pixels are then checked if they comply with any of the equations, where the number of pixels that fit each of the equations is stored. These numbers are know as Hough peaks, which can be visualized in a $\theta - \rho$ heat map, where ρ is the distance to the central pixel from an origin, as seen in figure 2.

$$xcos(\theta) + ysin(\theta) = \rho \tag{4}$$

It is noted that equation 4 is a straight line equation which allows a mathematical representation of a vertical line [8]. Figure 3 visualizes the result of a Hough transform, where the green squares visualize the highest peaks that determine the mostly likely parameters for a line in an image. In this example, it can be seen that two sets of parallel lines are detected, one at $\theta = 0$ and the other at $\theta = 50$.

III. Implementation

A. State machine

All the states are executed concurrently. The state machine selects which behavior takes the control of the drone. The procedure is straight forward, in the beginning the drone takes off and hovers at the desired altitude of 1.6m above the ground. Afterwards the line following is executed until the landing pad is detected. The landing procedure then takes over and is

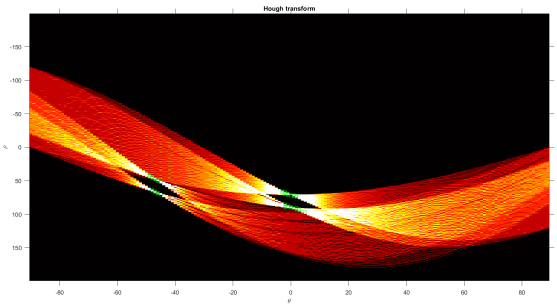

Fig. 3. Hough transform, green visualizes the lines

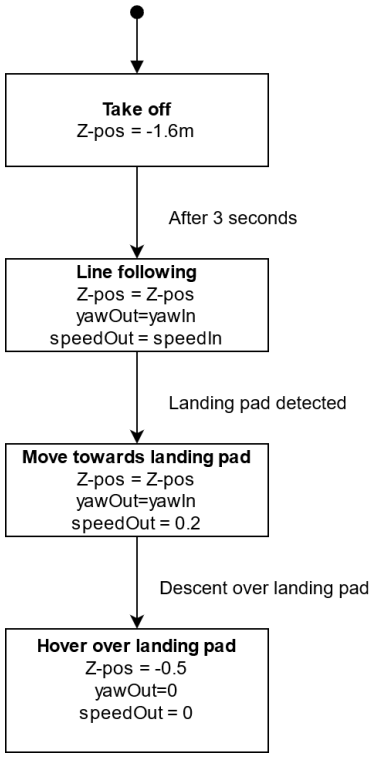

Fig. 4. State machine

executed until the distance between the drone and the center of the landing pad is below a threshold. The drone starts to descent and hovers 0.5m above the landing pad. The state machine is visualized in figure 4.

B. Image mask

To determine a mask which extracts the red path of the image, the MATLAB tool Color Thresholder is used. It allows the user to segment color images by thresholding color channels. The color channels RGB, HSV, YCbCR and L*a*b are supported. [9] It provides an export function, where the resulting MATLAB script is supported by code generation.

Since the task is to follow the red line, the color space RGB is chosen. The threshold is determined by using multiple images with different lightning conditions of a test track in real life. The result is visualized in figure 11.

C. Pipeline

This section describes the steps which are performed during the image processing, starting at the raw capture until the waypoint are calculated and passed to the controller. The intermediate results are visualized in the appendix.

Mask out center line The image is passed as RGB image from the Parrot minidrone. The image in simulation is visualized in figure 10. The edge detection requires a gray scale or single channel image. A color filter is applied, which is generated by the MATLAB tool Color Thresholder [9]. The result is a single channel image, see figure 11.

Create a skeleton In simulation, the path is by definition by the MathWorks team with a thickness of 10 cm [10]. As described in section II-D, the Hough transform detects edges. To create a edge at the center of the path, a skeleton is created, which reduces all objects to lines in a 2-D binary image based on their geometry, see figure 12.

Hough transform The Hough transform returns θ and ρ of where it thinks lines are likely to be, along with their start and end location. The result is a list of partial line segments which is visualized in figure 13.

Cluster line segments In this step the lines are clustered in multiple subsets, depending on the angle (θ).

Combine lines For each cluster, the overall length is calculated by taking the start and end position. The result is a smaller set of larger lines.

Attach lines The return value of the previous function is the start and end point of each large line. In this steps the lines are attached where the shortest distance between the end points of the lines are.

Calculate waypoints The line is represented as a start and end point. The waypoints are then all the points in between. Therefore linear interpolation is done between the start and end point of each line. The result is plotted in figure 14

Determine waypoints in front The image from the drone is an image from the top. Since the image contains portions of the track in front and behind the drone, we filter out points that are behind the drone. To do this, only the waypoints which are nearby the current drone position are taken into account. The list of nearby waypoints is split at the drone's position into two sublists. For each point of each sublist, the angle to the drone's position is calculated and averaged. The sublist with the closer average heading to the current drone heading is used and concatenated with the full list of waypoints and is then passed to the control system.

D. Trajectory Generation

A trajectory is generated from every incoming image following the waypoint extraction from the image processing. Using the two waypoints furthest along the track, as determined in the image processing, a third order polynomial is

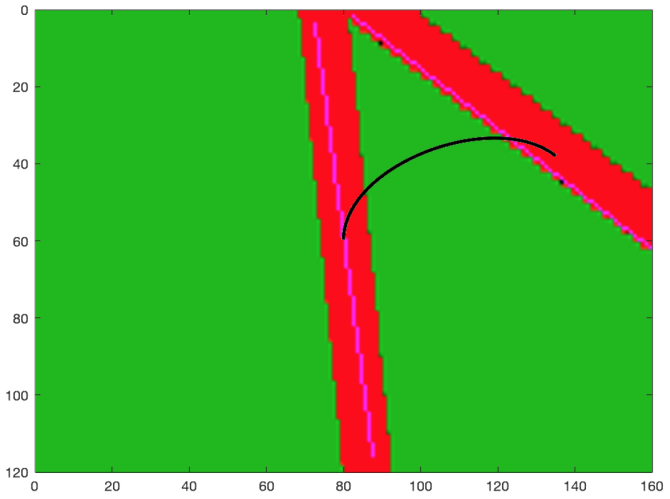

Fig. 5. View from the drone's camera with the generated trajectory overlaid in black. The pink lines are the extracted lines from the image processing.

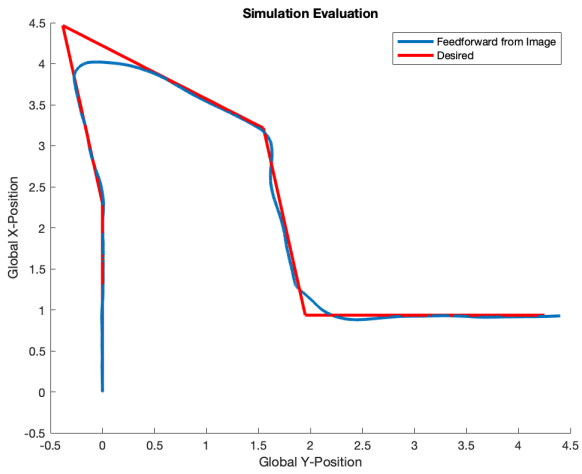

Fig. 6. Desired and recorded trajectories on a test track.

]

fit the position of the drone at the center of the image to the points. To determine the position of the apex of the corner the initial and final velocities in both X and Y directions are included in the polynomial fitting function. The initial velocity is know, since it is always along the X-axis of the body frame, where a constant value of 0.3 $[\frac{m}{s}]$ is used. The final velocity direction was determined from the directional difference between the two waypoints, and the magnitude was adjusted depending on if the drone's heading straight or turning. The adjustment is required since each trajectory is parameterized to have 200 time steps and straight trajectories required less time to complete and would result in fits with cusps, loops or S-shapes, all of which are not desired. This resulted in curves having a final velocity of 0.3 $[\frac{m}{s}]$, as seen in figure 5, while straight lines were reduced by a factor of ten to 0.03 $[\frac{m}{s}]$.

E. Controls

The controls were calculated using the trajectory and the equations in 3. While the trajectory was created with 200 time steps which totals 1 second, a new one was generated every 0.2 of a second. This means that only the first 40 steps are executed before a new trajectory is calculated. This results in real-time control adjustment and path correction while moving.

F. Velocity controller

The differential flatness controller returns a trajectory of the path, as well as speed. Due the fact that the execution time for each trajectory is the same, the drone has to increase the speed for a longer, and more complex trajectories. This results in a speed-up near the apex of corners and a reduction in speed for a straight path following. This behaviour causes issues for our model as the assumption for velocity heading matching the actual orientation is violated. Therefore, the speed controller is adjusted to invert this behaviour by subtracting the calculated value from a specified maximum. In figure 7, the output of the differential flatness controller with the inverted speed controller is plotted over the entire execution time of simulation seen in figure 6.

G. Landing

The target is a circular landing pad, which is placed at the end of the track, a short distance after the final line segment [10]. The landing pad detection behaviour searches for circles of a certain size within the image. A function is use to estimate the perimeter and area of circles in the image, which can be used to calculate the roundness of detected shape, using equation 5. Comparing the detected area to the known area with a diameter of $20cm$, and having a threshold for roundness, false positive detections can be filtered out. When a circle that meets the criteria is found, the centroid, which is also found from the function, is used as the next waypoint for the trajectory generation. The drone then starts descending if the centroid of the landing position is within $5cm$.

$$metric = 4\pi area/perimeter^2 \tag{5}$$

IV. Results

A. Simulation

To evaluate the proposed methods, testing was conducted on courses designed in the 3D World Editor from MathWorks. A simple single 90 degree corner test track is provided with the project files. This and other user defined tracks are used to test and incrementally improve the method. Figure 6 contains a plot of the most complex test map showing the desired track as well as the measured position. [1] The control inputs from the test are also shown in figure 7.

B. Physical experiment

For the physical experiment the Simulink Support Package for Parrot Minidrones is used, where algorithms can be

[1] A video of the test can be seen here: https://youtu.be/kNiId2MReso

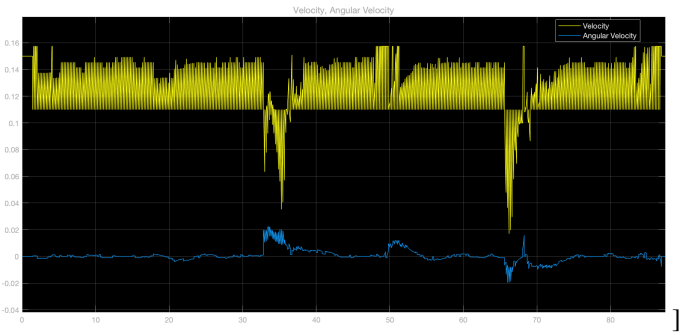

Fig. 7. Desired speed and angular velocity controls applied during the track in figure 6

deployed wirelessly over Bluetooth and executed on the drone [11], where a live video stream is supported.

The setup of the physical experiment is visualized in figure 8. The thickness of the path is slightly different from the simulation, which can be neglected due to the design of the image processing algorithm. The result of the image processing is visualized in figure 9 Further information and interpretation of the results are discussed in section V.

V. Discussion

From figure 6 it can be seen that our method perform well. In the shallow corners it has minimal overshoot and doesn't has significant oscillation either. While the sharper corners appear worse due to the amount of cutting, this is expected and acceptable as the next section comes into view and trajectories are updated. Figure 7, displays the rate at which the controls are being adjusted and a few turns can be seen directly. Our speed control can be seen to be working correctly, where the drone has its speed heavily throttled down during turns at 33 and 65 seconds. This is in contrast to the slight speed decreases at the first 10 degree turn after about 17 seconds and similarly at about 52 seconds during the second 10 degree turn. During the last sharp turn, which is executed after 68 seconds, a spike can be seen in the velocity controller. It turned out that the spike results after one line segment was not extracted by the image processing algorithm. This causes a rapid jerk in the heading of the drone but since it is only a single frame the drone is able to recover and complete the course.

During the physical experiment the drone started to shake and behave unexpectedly. We then proceeded to debug the issue by providing constant control values, which should lead the drone to a stable hovering. This lead to the same result, with the drone still unable to hover in a controlled manner and therefore, no successful experiment could be executed. This is thought to be caused by the Parrot Mambo's on board ARM-9 800MHz single core processor [12] not being powerful enough for our image processing algorithm, which is for comparison less than a Raspberry Pi Zero, which has a clock frequency up to 1 GHz [13]. This was determined by using a Simulink

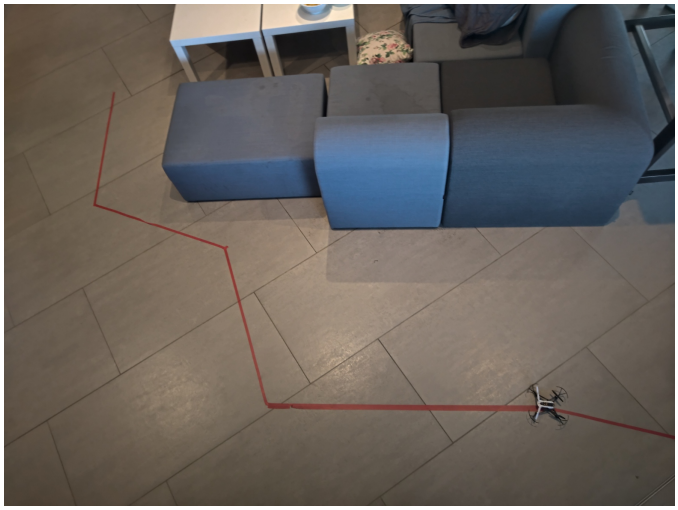

Fig. 8. Setup of physical experiment

model, from the Parrot Minidrone package, that isolates the image processing model. After loading this model onto the drone with our algorithm we were able to see the results in 9. This model produced the expected results since there is no computational power going towards the control on the drone. We believe the image processing is taking so long that the drone is unable to update even its lower level controls.

VI. Conclusion

This paper describes a real-time image-based feedforward control systems for a standard quadcopter. Using a constrained dynamical model, we were able to produce an autonomous navigation method capable of completing a track using a line following method, in a simulation with high precision. When deploying our solution to hardware we were met with unexpected challenges involving hardware limitations. This limited our results, by only being able to prove our image processing could work on the physical drone without having to provide any control. Future work on this topic could include deploying our solution onto another, more powerful drone such as the Parrot Bebop 2, which is also supported by the Mathworks hardware package. Finally we would look to expand our control to include more degree's of freedom to enable more accurate and complex trajectories.

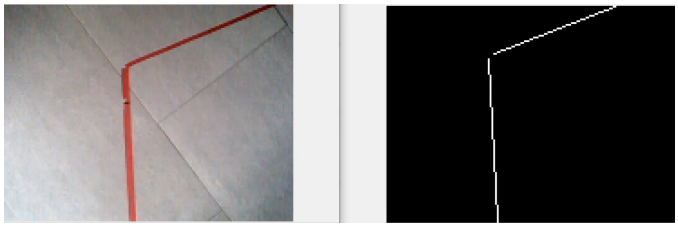

Fig. 9. Experimental video processing, left: RGB image, right: extracted line after image processing

References

[1] "Market revenue of drones worldwide 2019-2025," [Online]. Available: https://www.statista.com/statistics/1200348/drone-market-revenue-worldwide/.

[2] "Lobal autonomous bvlos drones market (2020 to 2025) - growth, trends, and forecasts," 2020.

[3] "European atm master plan - roadmap for the safe integration of drones into all classes of airspace," 2018. [Online]. Available: https://www.sesarju.eu/sites/default/files/documents/reports/European%20ATM%20Master%20Plan%20Drone%20roadmap.pdf.

[4] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[5] R. Kumar, A. Nemati, M. Kumar, R. Sharma, K. Cohen, and F. Cazaurang, "Tilting-rotor quadcopter for aggressive flight maneuvers using differential flatness based flight controller," in *Dynamic Systems and Control Conference*, American Society of Mechanical Engineers, vol. 58295, 2017, V003T39A006.

[6] M. Islam, M. Okasha, and M. M. Idres, "Dynamics and control of quadcopter using linear model predictive control approach," *IOP Conference Series: Materials Science and Engineering*, vol. 270, p. 012 007, Dec. 2017. DOI: 10.1088/1757-899x/270/1/012007. [Online]. Available: https://doi.org/10.1088/1757-899x/270/1/012007.

[7] I. Haq, S. Anwar, K. Shah, M. T. Khan, and S. A. Shah, "Fuzzy logic based edge detection in smooth and noisy clinical images," *PloS one*, vol. 10, no. 9, e0138712, 2015.

[8] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[9] *Mathworks color threshold*, https://de.mathworks.com/help/images/ref/colorthresholder-app.html, Accessed: 2021-06-03.

[10] *Mathworks minidrone competition - rules and guidelines*, https://se.mathworks.com/content/dam/mathworks/mathworks-dot-com/academia/student-competitions/minidrone-competition/mathworks-minidrone-competition-guidelines.pdf, Accessed: 2021-10-06.

[11] *Simulink support package for parrot minidrones*, urlhttps://se.mathworks.com/matlabcentral/fileexchange/63318-simulink-support-package-for-parrot-minidrones, Accessed: 2021-12-06.

[12] *Parrot mambo sip 6 linux motherboard with 800 mhz arm a9*, https://www.bhphotovideo.com/c/product/1274651-REG/parrot_pf070237_mambo_sip6_linux_motherboard.html, Accessed: 2021-12-06.

[13] *Raspberry pi zero*, https://www.raspberrypi.org/products/raspberry-pi-zero/, Accessed: 2021-12-06.

APPENDIX

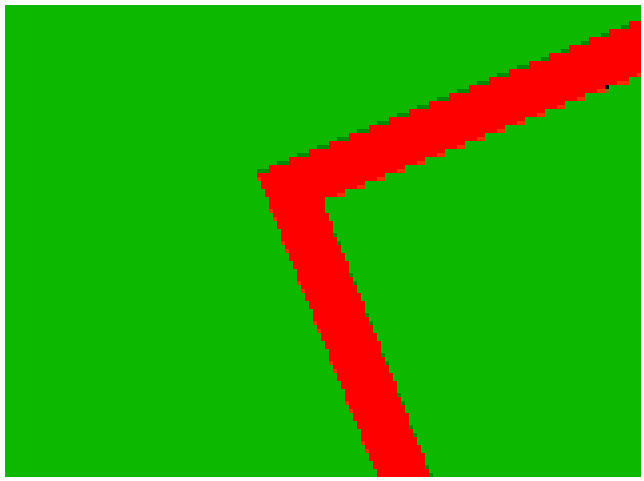

Fig. 10. Example track in simulation

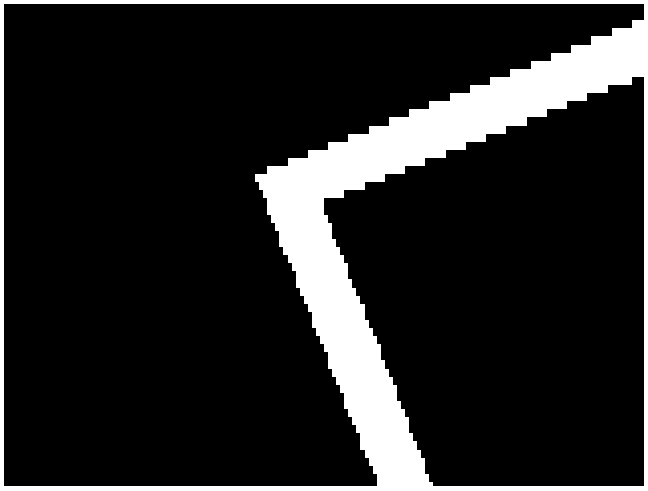

Fig. 11. BW image of track

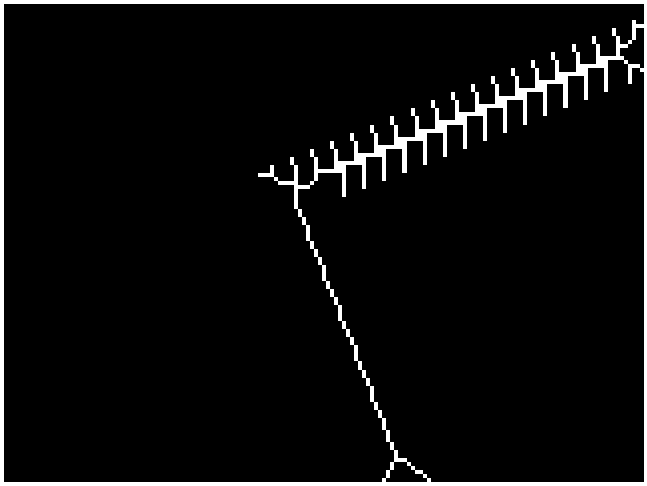

Fig. 12. Skeleton

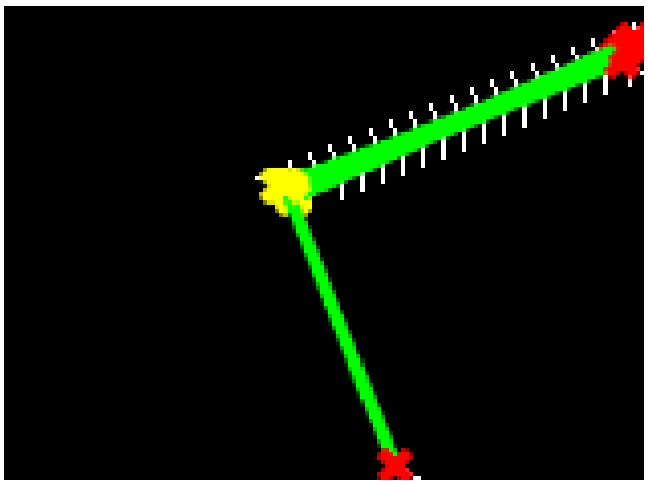

Fig. 13. Hough transform

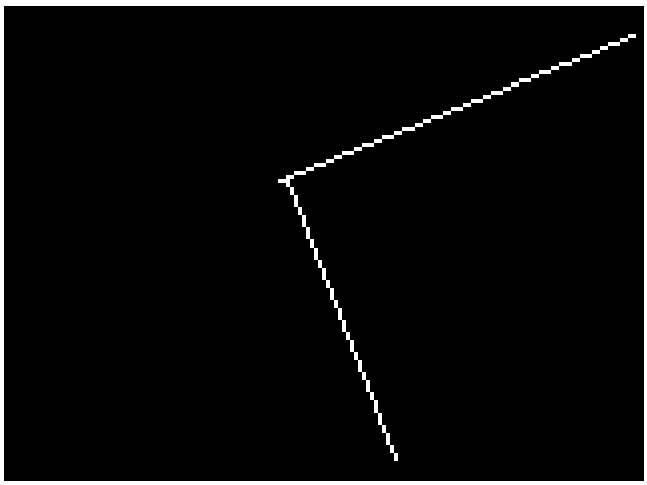

Fig. 14. Final result - Extracted track