Term Project: Air supported Search and Rescue

Albin Frischenschlager, 0926427
Markus Hennebach, 0926654
Patrick Knöbel, 0925219

1 Introduction

The need to find missing objects or persons within a given time in order to ensure their liveness has always been a cost intensive task. This is due to the large effort of personnel and the special equipment needs induced by a rough operation environment. Searching persons in mountainous area for instance especially relies on the support of air units like helicopters. However, the usage of smaller and more efficient unmanned aerial vehicles (UAVs) leads to a less cost-intensive ressource usage and hence to a faster searching when more units are deployed.

1.1 Specification

The UAV is provided with two lengths and a GPS coordinate describing the center, the height and the width of the rectangle of the area to be searched the center GPS coordinate of the rectangle-shaped target area in which the searched object is supposed to be located. Initially, the drone starts in a steady state outside of the described target area. The system is then commanded to take off and flight towards the first corner position at a specified height. After matching the first corner at a precision of 2.5m, the UAV heads towards the second corner point. When arriving at the second point the first two distances are redefined and the drone gets the third corner point as its next target. This procedure is repeated until the target's coordinate is reached or until the entire region has been searched. The device ultimately stops at this last position. The detection of the searched object is modeled as follows: When the drone spots the object in its "visual range" (about 15m), the current coordinate is saved and the drone stops there.

2 Concept and Background

2.1 Guidance system

In order to make the UAV able to approach a GPS coordinate, a guidance system is necessary. Many different approaches exist to solve this problem. In this project's final version the authors opted for the option to use the north pole as a static reference point. Other approaches will be discussed later on in section 3.

The guidance system's principle:

- First step is to adjust the drone's orientation towards the north pole. For this purpose a compass is requited.
- During the guidance the static referenced angle between the actual position of the drone and the target position is calculated.
- Finally the drone is accelerated in the direction of the calculated angle.

When the UAV is approaches the target (at an offset of 10m) the speed is reduced in order to stop the drone at the target location. For more detailed explanations read section 4.5.

2.2 Calculating distance and angle between two GPS coordinates

The distance between two coordinates can be calculated using the formulas describe in[3]:

$$t = \sin\left(\frac{T_{lat} - P_{lat}}{2}\right)^2 + \cos(P_{lat})\cos(T_{lat})\sin\left(\frac{T_{lon} - P_{lon}}{2}\right)^2 \tag{1}$$

$$distance = 2R\ \arctan\left(\frac{\sqrt{t}}{\sqrt{1-t}}\right) \tag{2}$$

The angle between two positions can be calculate as following [2]:

$$y = \sin(T_{lon} - P_{lon})\cos(T_{lat}) \tag{3}$$

$$x = \cos(P_{lat})\sin(T_{lat}) - \sin(P_{lat})\cos(T_{lat})\cos(T_{lon} - P_{lon}) \tag{4}$$

$$angle = \arctan\left(\frac{y}{x}\right) \tag{5}$$

Where P is the current position and T is the target position.

For the correct computation of *angle* the correct quadrant for the result of $\arctan(y/x)$ has to be involved. In MATLAB the function $arctan2(y, x)$ provides the feature.

2.3 Ziegler–Nichols method

The Ziegler–Nichols method is a heuristic method for tuning a PID controller [1]. This method is optimized to be used in real world applications.
It includes the following steps:

- Turn off all parameters (P, I and D)
- Increase the parameter P until the system starts to oscillate

This parameter is also denoted as the ultimate gain K_u. The period of oscillation is called P_u. To calculate the parameters of the PID controller the following formulas are used:

$$P = 0.60 \cdot K_u \tag{6}$$

$$I = \frac{2 \cdot P}{P_u} \tag{7}$$

$$D = \frac{P \cdot P_u}{8} \tag{8}$$

3 Misconceptions

3.1 Guidance system

Another approach to the final one has been a developed but due a certain number of difficulties not longer pursued. Its basics are summarized in the following points:

- At first, the UAV is accelerates forwards.
- Then, the relative speed of the GPS is used to calculate the current course.
- Furthermore, the angle from the current position to the target position is calculated.
- This information is continuously used to correct the direction of the drone to point at the target.

When the drone closes up the target, the speed is curbed. A problem occurs when the speed undercuts a certain speed, the calculation of the flying direction is then getting inaccurate. This leads to incorrectable problems. The drone will therefore have a permanent offset to the target.

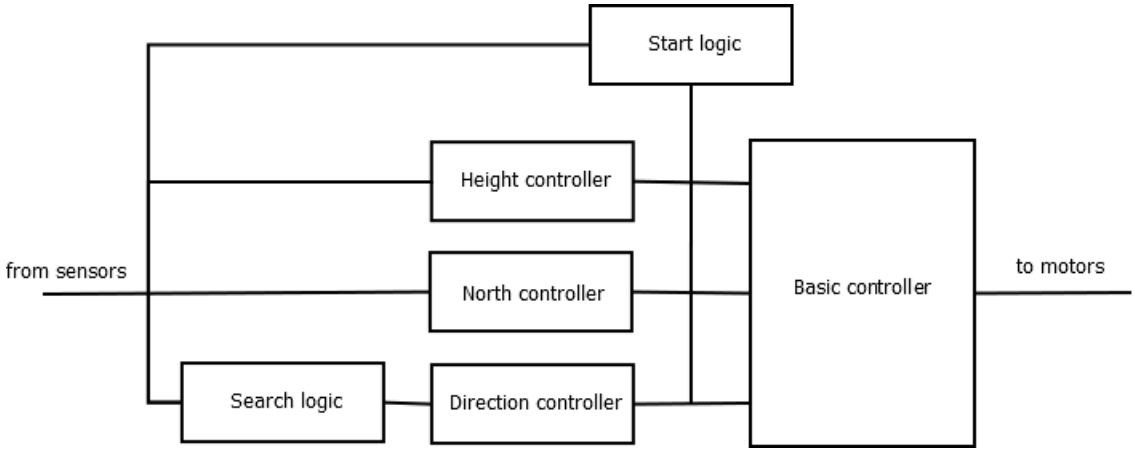

Figure 1: Overview

4 Implementation

The high level controller can be split in six parts shown in figure 1. Each part will be described more in detail hereafter.

4.1 Starting sequence

To make the high level controller more reliable, this safety system is included. For this complex and sequential task a *Stateflow* – shown in figure 2 – is used.

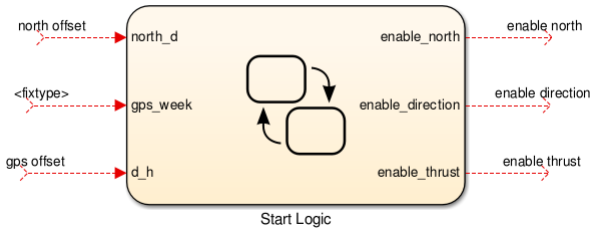

Figure 2: Start logic block

The inputs of the block are used to monitor the state of the drone. Figure 3 shows the states of the Block. The outputs are used to turn the subsystems on and off.

States:

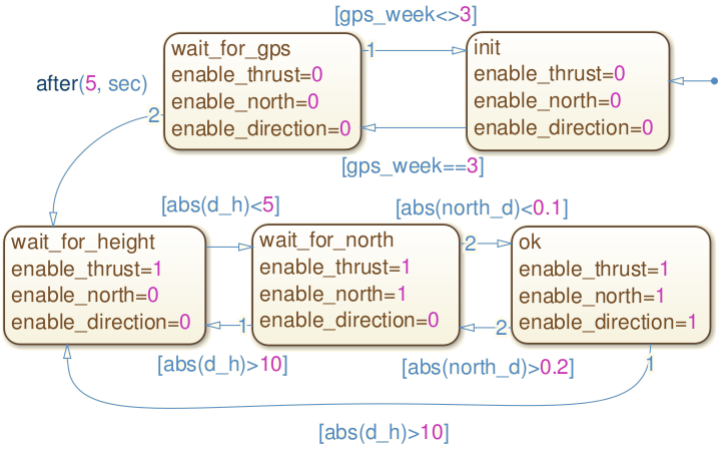

Figure 3: Start logic

- *init*:
  This state is entered when the high level controller is started. When the GPS signal is locked the next state will be entered. In this state all subsystems are turned off.

- *wait_for_gps*:
  This state ensure that the GPS signal is stable for 5 seconds. If not, it jumps back to the *init* state.

- *wait_for_height*:
  If the GPS signal has been stable for 5 seconds this state is entered. In this state the height controller is enabled.

- *wait_for_north*:
  If the drone's height surpasses 5 meters, this state is entered. A jump back to *wait_for_height* occurs, if the height underpasses 10 meters. In this state the north controller is also enabled.

- *ok*:
  As long as the height is bigger than 5 meters and the orientation towards the north differ by less than 0.1 radiant, this state is entered. Again, jump backs will occur if one of the conditions is not satisfied. In this state all subsystems are enabled.

4.2 Basic controller

The basic controller stabilizes the drone. All parameters have been taken form the low level controller *Attitude_Contoler*, however the design is morphed. Figure 4 shows the block diagram of the basic controller.

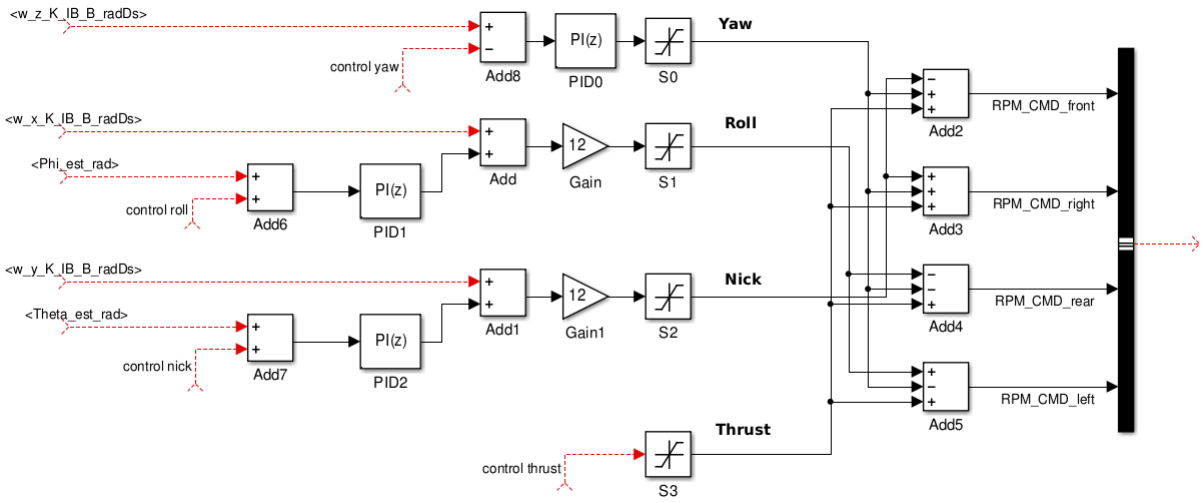

Figure 4: Basic controller

The inputs <*w_x_K_IB_B_radDs*>, <*w_y_K_IB_B_radDs*>, <*w_z_K_IB_B_radDs*>, <*Phi_est_rad*>, <*Theta_est_rad*> are used to stabilize the drone the same way as the *Attitude_Contoler* does. The other inputs are used to manipulate the position of rest:

- *control yaw*:
  This input controls the rotational speed [radiant/s] (called yaw) of the drone. If the input is zero then the basic controller stops the rotation of the UAV.

- *control roll* and *control nick*:
  These two inputs are used to adjust the *pitch* of the drone in radiant. If they are both zero, the basic controller keeps the drone horizontal.

- *control thrust*:
  The thrust input is an absolute value and should be between 40 and 100.

4.3 Height controller

Figure 5 shows the block diagram of the height controller. The input signal <*h_R_MSL_m*> from the GPS receiver is used to get the drones current height.

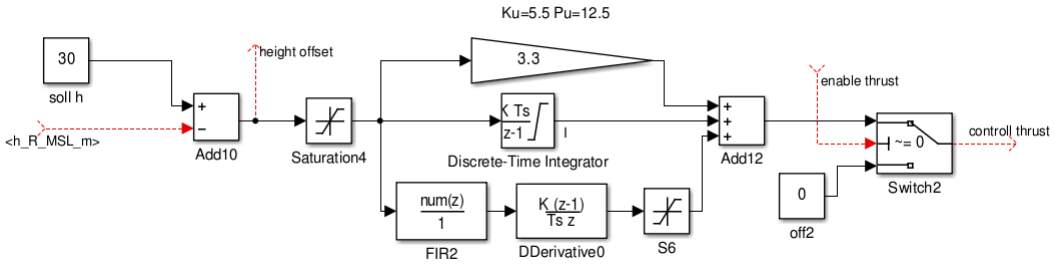

Figure 5: Height controller

With the constant *soll h* the height can be adjusted.

To control the height a PID controller is used. The controller is parametrized using the *Ziegler–Nichols* method (section 2.3). It started oscillating at a gain of 5.5 and an oscillation period of 12.5 seconds.

The output *control thrust* is directly connected to the basic controller. The output *height offset* and the input *enable thrust* are connected with the start logic. Therefore, *Switch2* is used to enable the height controller.

4.4 North controller

For our guidance system we need to turn the drone in the direction of the north pole. This controller is shown in figure 6. The inputs <*B_x_R_B_T*> and

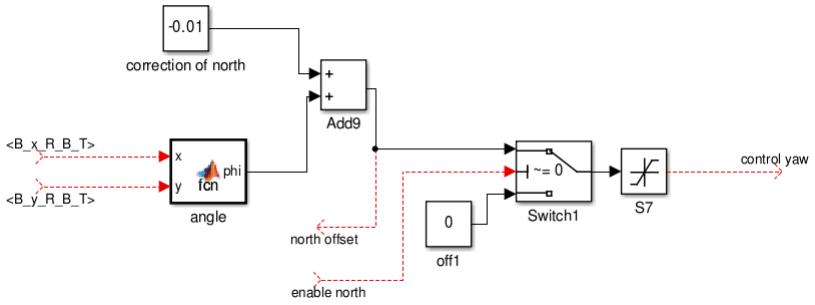

Figure 6: North controller

<*B_y_R_B_T*> from the magnetic field sensor are used to determine the bias of the drone. The function block *angle* converts this inputs to a static referenced angle:

```
function phi = fcn(x,y)
```

```
    phi = atan2(y,x);
end
```

If the calculated output is zero it means that the drone is exactly pointing to the north pole. If the output *phi* is grater then zero the drown has to turn. Therefore the output of the block is directly used for the basic controller to manipulate the yaw of the drone. The limiter *S7* is used to limit the rotational speed of the drone. This makes the system more stable.

The input *enable north* and the output *north offset* is connected with the start logic. Therefore *Switch1* is used to turn of the noth controller.

We had a problem with the derivative part of the controller. The GPS signal is only refreshed every second but the simulation is running with 1000 samples per second. Therefore the derivative part of the controller only produces spikes. Our solution is to use s FIR filter with makes a simple averaging over 1000 samples.

4.5 Direction controller

Figure 7 shows the block diagram of the direction controller. The controller

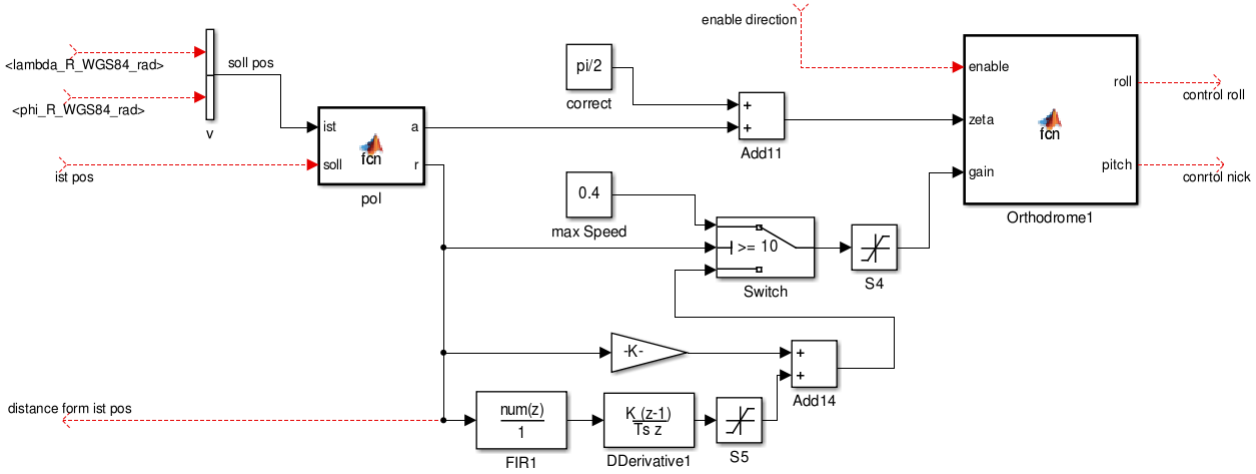

Figure 7: Direction controller

has two inputs *ist pos* (the current GPS position of the drone) and *soll pos* (the target position). The function block *pol* converts the two positions into a distance r and an static referenced angle a:

```
function [a,r] = fcn(ist,soll)
    clng=ist(1);
```

```
    clat=ist(2);
    dlng=soll(1);
    dlat=soll(2);

    temp = (sin((dlat - clat)/2))^2 + cos(clat) *
            cos(dlat) * (sin((dlng - clng)/2))^2;
    r = 6378140 * 2 * atan2(sqrt(temp), sqrt(1-temp));

    a = atan2(sin(dlng-clng)*cos(dlat),cos(clat)*sin(dlat)-
        sin(clat)*cos(dlat)*cos(dlng-clng));
    %Formulas from Dr. Math
    %http://mathforum.org/library/drmath/view/51879.html
    %http://mathforum.org/library/drmath/view/55417.html
end
```

The angle is then used directly to head for the target direction. To convert the angle to the static reference angle of the north controller we have to add $\pi/2$ to it.

The usage of the distance r is more complicated. If the distance is higher then 10 meters the drone is flaying at the max speed. Therefore the *Switch* and the constant *max Speed* is used. If the distance to the target is less then 10 meter a PD controller is used. It is not necessary to use an integral part, because the drone itself acts like an integrator. Furthermore the controller has an absolute value as input. This makes the control path non-linear. Therefore it is not possible to use the *Ziegler–Nichols* method for tuning. We tuned the controller manually. The derivative parameter is chosen in a way that the drone slow down very quickly when it reached the target. The proportional parameter is chosen in a way that the drone is stable. If it is to high the drone starts to oscillate. With the derivative part of the controller we had the same problem like in section 4.4. Over all the limiter *S4* is used to limit the maximum speed of the drone to make the system more stable.

The input *enable direction* is coming from the start logic to enable or disable the direction controller. The function block *Orthodrrome1* is used to calculate the static angle of roll and nick for the basic controller:

```
function [roll, pitch] = fcn(enable,zeta,gain)
    if enable == 0
        gain=0;
    end
    pitch = sin(zeta)*gain;
    roll = cos(zeta)*gain;
end
```

The output *distance from ist pos* is connected to the search logic.

4.6 Sensor

The sensor for rescue searching is emulated. Figure 8 shows the block diagram of the emulator. The function block *scanner* calculates the distance between

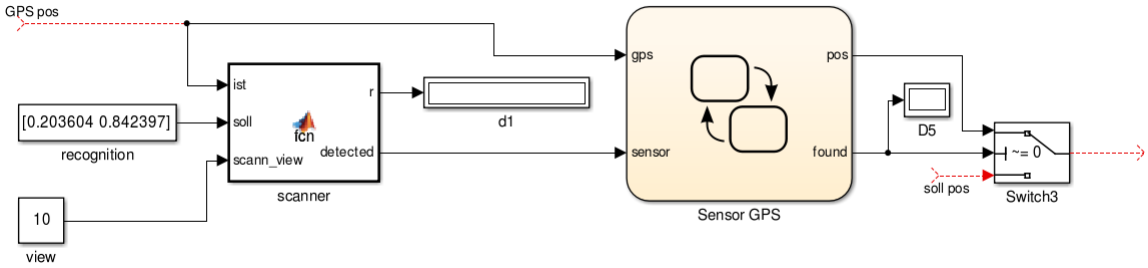

Figure 8: Sensor emulation

the virtual *recognition* point and the current position of the done. If the drones distance is less then 15 meters the output *detected* is turn on. The stateflow

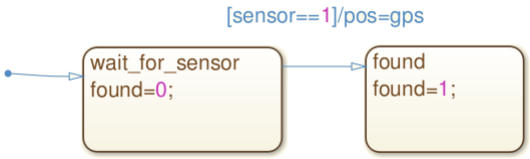

Figure 9: Sensor logic

graph is shown in figure 9. It very simple saves the current position of the drone if the emulated scanner detects something. Finally *Switch3* is used to switch between the *soll pos* position form the search logic and the saved position. The output of the switch is directly connected to the direction controller. Therefore if the scanner triggers the drone will just hold the current position.

4.7 Search logic

The stateflow block shown in figure 10 is the main search algorithm. The input *distance from ist pos* is used to determined if the drone has reached a target point. The output *soll pos* is directly connected to the direction controller over the switch of the sensor.

Figure 11 shows the stateflow diagram of the search logic. There are two function used. The function *decrease*, which simply decrease the input value *vin*

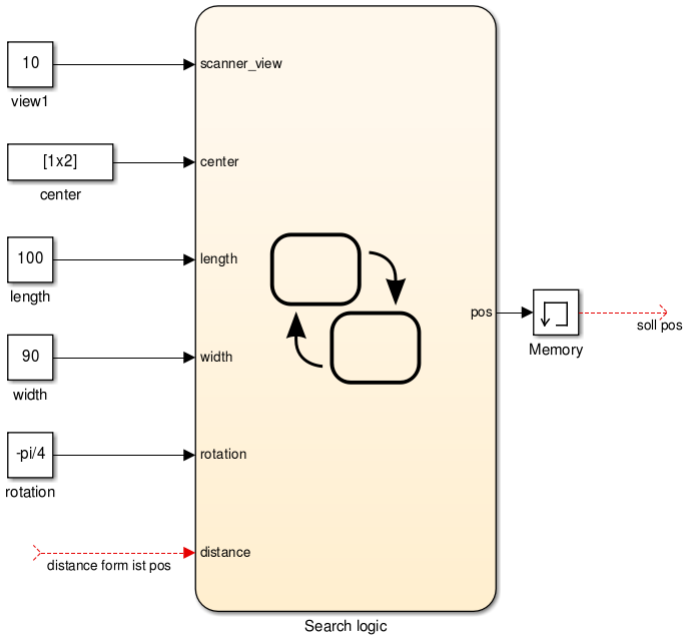

Figure 10: Search logic block

with the value of *step*. If the calculated value is less zero then zero is returned. The other function *getpoint* calculates a new GPS coordinate. Therefore you have to give a GPS coordinate, a static referenced angle and a distance:

```
function pout=getpoint(pin,brng,r)
    lat1=pin(2);
    lon1=pin(1);
    pout=pin;
    lat2 = asin( sin(lat1) *     cos (r/6378140) + cos(lat1) *
           sin(r/6378140) * cos(brng) );
    lon2 = lon1 + atan2( sin(brng) * sin(r/6378140) * cos(lat1),
           cos(r/6378140) - sin(lat1) * sin(lat2) );
    pout(2)=lat2;
    pout(1)=lon2;
    % http://www.movable-type.co.uk/scripts/latlong.html
end
```

States:

- *init*:
  In the *init* state all local variable are set. Also the start pos is calculated.

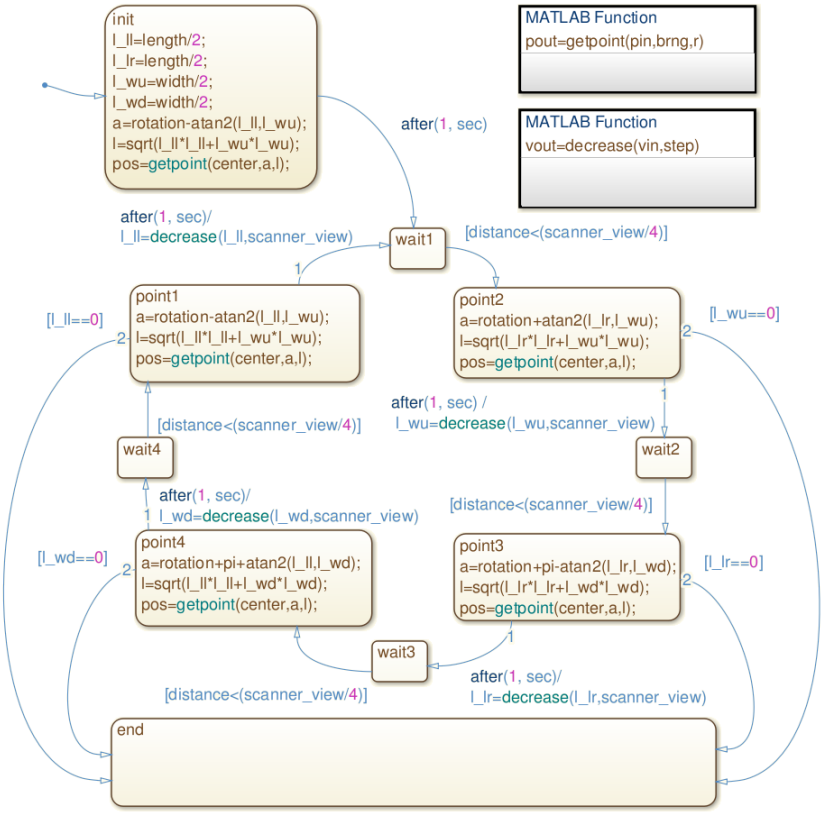

Figure 11: Search logic

- *wait1-4*:
  In the *wait* state the search logic is waiting for the drone to reach the target position. If the distance is less then 2.5 meter then it jumps to the next *point* state.

- *point1-4*:
  In this state the new target position is calculated. It will jump to the wait state after the calculation is done. During the jump also the corresponding local variable for the corner is decreased.

- *end*:
  If there is no space between a corner point an the center point left the *point* states will jump here. This is the end point and the drone will stop searching.

5 Conclusion

We achieved good results with this solutions. Figure 12 shows the position of

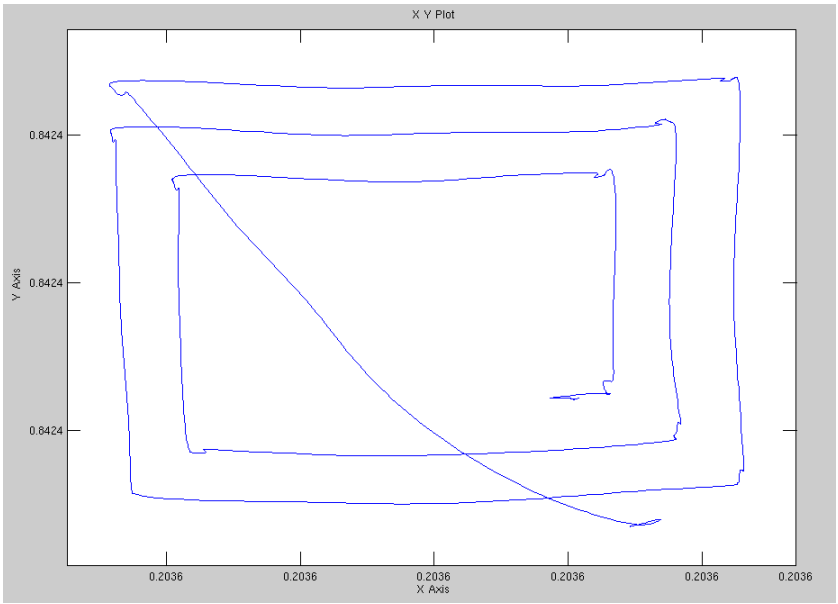

Figure 12: X-Y graph of the drone position

the drone in a X-Y graph. The drone sometimes needs to change the direction to hit a target point. This can be optimized, but it also shows that the guidance-system is working really stable.

A week point of this implementation is the dependency of a strong GPS signal. In the real word is can happen that the signal gets weak or will be lost completely. To reduce the risk of failure it is necessary to use sensor fusion or to build a backup system.

A Effort

Task	Frischenschlager	Hennebach	Knöbel
Basic controller	11	11	8
Start logic	0	0	3
Height controller	6	6	6
North controller	8	6	0
Direction controller	6	6	6
Search logic	0	0	6
Protocol	0	6	16
Sum	31	35	45

B References

References

[1] *Ziegler–nichols method*, `en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method`, [Online; accessed 2014].

[2] The Math Forum Doctor Rick, `mathforum.org/library/drmath/view/55417.html`, [Online; accessed 2014].

[3] Doctors Rick and The Math Forum Peterson, `mathforum.org/library/drmath/view/51879.html`, [Online; accessed 2014].