Towards Collision Avoidance for Commodity Hardware Quadcopters with Ultrasound Localization

Benjamin Vedder[1,3], Henrik Eriksson[1,4], Daniel Skarin[1,5],
Jonny Vinter[1,6], and Magnus Jonsson[2,7]

***Abstract*— We present a quadcopter platform built with commodity hardware that is able to do localization in GNSS-denied areas and avoid collisions by using a novel easy-to-setup and inexpensive ultrasound-localization system. We address the challenge to accurately estimate the copter's position and not hit any obstacles, including other, moving, quadcopters. The quadcopters avoid collisions by placing contours that represent risk around static and dynamic objects and acting if the risk contours overlap with ones own comfort zone. Position and velocity information is communicated between the copters to make them aware of each other. The shape and size of the risk contours are continuously updated based on the relative speed and distance to the obstacles and the current estimated localization accuracy. Thus, the collision-avoidance system is autonomous and only interferes with human or machine control of the quadcopter if the situation is hazardous. In the development of this platform we used our own simulation system using fault-injection (sensor faults, communication faults) together with automatically-generated tests to identify problematic scenarios for which the localization and risk contour parameters had to be adjusted. In the end, we were able to run thousands of simulations without any collisions, giving us confidence that also many real quadcopters can manoeuvre collision free in space-constrained GNSS-denied areas.**

I. Introduction

In order to test and demonstrate different applications on Micro Air Vehicles (MAVs), a platform that is easy to set-up and safe to operate can be very useful. We envision a quadcopter platform built from inexpensive hardware that can be set up at new locations in less than 15 minutes. Our targeted environments have constraints on space and lack of Global Navigation Satellite Systems (GNSSs), which makes it difficult to navigate autonomously compared to outdoor environments. This platform should give the pilot, who can be a human or a machine, full control in normal circumstances while preventing collisions when the situation gets hazardous, regardless of pilot input. Thus, the quadcopters have to be aware of their own positions and the positions of static and moving objects in the area. They also have to be aware of the accuracy of their position and the physics that restrict how they can manoeuvre.

Our platform is designed to meet the following requirements:

- No sensitivity to lighting conditions and background contrast, as is the case with camera-based systems [1]–[4].
- The computations for estimating the position and avoiding collisions should be inexpensive enough to be handled by on-board microcontrollers (as opposed to offloading them to external computers [3], [4]).
- The extra equipment on each quadcopter should be light enough to allow extra payload and spare the battery. There are solutions with relatively heavy laser range finders that do not meet this requirement [5], [6].
- There should be fault tolerance to e.g. handle occasional faulty distance measurements.
- Pilot errors should be handled by automatically taking over control if the situation becomes hazardous.

To meet these requirements, we have created a localization system that uses ultrasound to measure the distance between the copters and several station-

[1]Dept. of Electronics. SP Technical Research Institute of Sweden
[2]School of Information Science, Computer and Electrical Engineering. Halmstad University
[3]benjamin.vedder@sp.se
[4]henrik.eriksson@sp.se
[5]daniel.skarin@sp.se
[6]jonny.vinter@sp.se
[7]magnus.jonsson@hh.se

ary anchors. The ultrasound-localization hardware is based on open-source radio boards [7]. To make the copter's aware of each other, they communicate their positions and velocities to each other on a regular basis.

Testing the system has been a significant part of this work. We have developed a simulator that operates together with fault injection [8] and property-based testing [9] techniques to evaluate how a larger system with quadcopters behaves while hardware faults and/or pilot misbehaviour occurs. This way, we could randomly generate pilot control commands and inject faults during thousands of automatically generated simulations to see when a collision occurs. For fault injection, we used the FaultCheck tool [10] and for generating tests we used the Erlang QuickCheck tool [11]. When we had a sequence of pilot and fault injection commands that led to a collision, we used the shrinking feature of QuickCheck to get a shorter test sequence of commands that leads to a collision. We could then run this sequence of commands in the simulator repeatedly, while adjusting the system parameters, until it would not lead to a collision anymore.

Dealing with the slow update rate of the anchors, with the simulation-hardware relation, and with the occasional measurement faults of the system was challenging. Even so, we achieved a result with a functioning copter platform and much shared code between the simulator and the hardware.

The contributions of this work are the following:

- A novel hardware and software solution for doing localization in GNSS-denied areas based on ultrasound measurements fused with Inertial Measurement Unit (IMU) data using easily available, inexpensive hardware.
- A technique to take over control in hazardous situations to avoid collisions between moving quadcopters by using communication between them and risk contours.
- We show how performance and fault tolerance can be evaluated with automatically generated tests using our previously proposed platform that utilizes fault injection and property-based testing [10].

The rest of the paper is organized as follows. Section II presents related research, Section III describes our hardware platform, Section IV describes our ultrasound distance measurement technique and Section V shows how we do position estimation. Further, in Section VI we describe our collision-avoidance technique, Section VII describes our simulations and in Section VIII we present our conclusions from this work.

II. Related Work

Much research has been devoted to autonomous MAVs, such as quadcopters. Early systems worked only outdoors as they relied on GNSS positioning systems [12]. Recently, part of this research has been devoted platforms that operate in GPS-denied areas such as indoor environments [1]–[6], [13], [14]. One approach is to use cameras either mounted on the copters to identify the environment [2], [4], [13]; or external cameras that identify markers on the copters [1], [3]. Limitations with the camera-based solutions are that they require much computational power and good light/contrast conditions. Many camera-based solutions run the computation on a stationary computer and send the results back to the copter [1]–[3], [13], [14]. Another approach is to use laser range finders mounted on the copters to run Simultaneous Localization and Mapping (SLAM) algorithms [5], [6]. This approach often works without modifying the external environment with e.g. anchors or cameras, but relies on the environment having walls that are close enough to be detected. Limitations with laser range finders are that they are relatively expensive and quite heavy, adding much payload to the weight-constrained copter.

Similar to our platform, there is one early system that relies on infra-red and ultrasound sensors mounted on quadcopters that measure distances to walls and the floor [15]. These copters can avoid collisions, but did not have enough accuracy to perform a stable hover. More recently, a platform has been presented by J. Eckert that uses ultrasound localization with inexpensive hardware to manoeuvre quadcopters [14], [16], [17]. This platform uses a swarm of small robots that spread out on the floor and allow a consumer (a quadcopter in this case) to hover above them. Compared to our platform, Eckerts's ultrasound system has a shorter range, of 2 m when there is noise from quadcopters, while

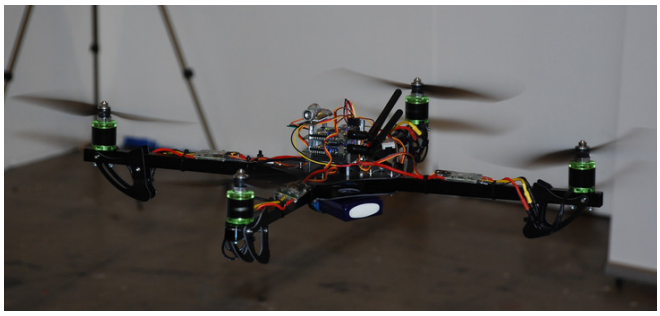

Fig. 1. A photo of one of our quadcopters.

Fig. 2. A photo of four ultrasound anchors mounted on tripods.

our system can operate at distance of up to 12 m from the anchors with the current configuration. Eckert's quadcopter also relies on optical flow sensors aimed towards the floor and ceiling because the update rate from their ultrasound system is too low and not as tightly coupled to the control loop as our system. Thus, their localization depends on having relatively good contrast and lightening conditions and a ceiling that is low enough, which makes it difficult to use outdoors.

To our knowledge, beside our quadcopter system, there is currently no other indoor quadcopter system that can do a stable hover and collision avoidance with only ultrasound localization and IMU-based dead reckoning. Our system also has a unique approach on collision avoidance and fault tolerance.

III. Hardware Setup

Our platform consists of several quadcopters and several (at least two) stationary anchors, as shown in Figure 3. A photo of one of our quadcopters is shown in Figure 1 and a photo of the anchors is shown in Figure 2. The anchors and quadcopters have synchronized clocks to do Time of Flight (ToF) measurement of ultrasound to determine the distance between them. The copters also have one ultrasound sensor each that measures the distance to the floor. Since the $[x, y, z]^T$ position of the anchors is known by the copters, they can calculate their own position based on the distances to the anchors. Two anchors are enough for this system to work if the copters never pass the line between the anchors, but any number of anchors can be used to provide more accuracy and/or redundancy.

A block diagram of the hardware components on each quadcopter and their connections can be seen in Figure 4. There is one custom main

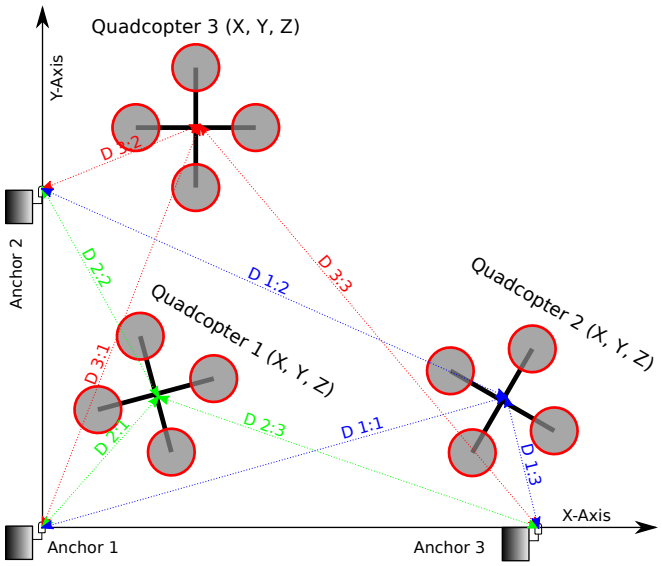

Fig. 3. The ultrasound-localization system with anchors. The measured distances from the copters to the anchors are marked D a:b.

controller board that is responsible for the high-speed (1000 Hz) attitude control loop. The position control loop and part of the position estimation is also done here. The components and their functions on the mainboard are:

- The STM32F4 microcontroller is responsible for all computation and communicates with the other components on the mainboard.
- The MPU9150 IMU sensor provides the raw data that is used for attitude estimation. It has a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer; thereby providing nine degrees of freedom.
- The barometer measures the air pressure and is currently not used in any algorithm. Later, it could be used for redundancy when measuring

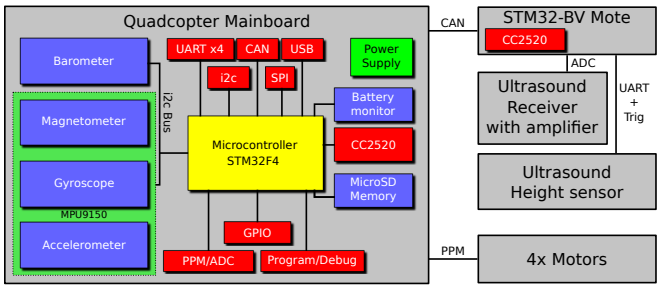

Fig. 4. The hardware setup on each quadcopter.

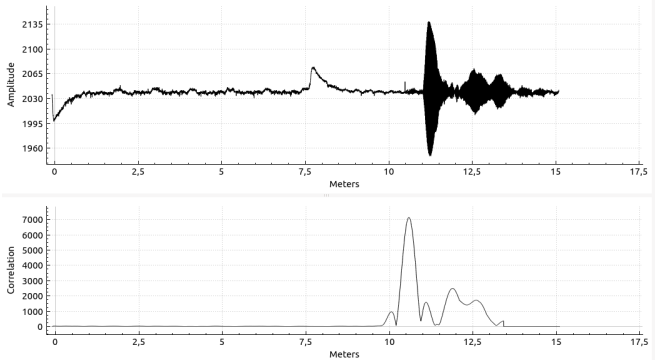

Fig. 5. Ultrasound samples recorded for a time corresponding to 10 meters and the cross correlation result.

the altitude.

- The CC2520 radio transceiver is used to communicate with the ground station and other quadcopters.
- Standard Pulse-Position Modulation (PPM) signals are sent to motor controllers that drive the propeller motors.

The STM32BV-mote is responsible for clock synchronization and distance measuring. An ultrasound receiver is connected to an Analog to Digital Converter (ADC) pin with a simple amplifier to capture pulses from the anchors. The mote also communicates with an ultrasound altitude sensor to measure the distance to the floor. These measurements together are sent to the mainboard that computes the position of the copter based on them.

IV. Ultrasound Distance Measurement

The ultrasound distance measurements are done by synchronizing the clocks of all anchors and quadcopters, and having timeslots assigned when pulses are sent from different anchors. The pulses are then recorded by the receivers on the quadcopters during these timeslots. The receiver then uses the ToF of the pulse to calculate the distance to the anchor.

Clock synchronization is done by having a node sending out a clock value and using a hardware interrupt on the receiving nodes that saves the local clock value at the time the packet starts being received. When the whole clock packet is received, the difference between the time stamp and the received clock value is subtracted from the own clock. This difference is also used to estimate the clock drift and compensate for that over time. With clock packets sent every 2 s, the clock has a jitter of less than 5 µs, which is good enough to measure the ToF of sound.

In order to reject noise on the ultrasound measurements, the pulses sent out by the anchors are created by multiplying the 40 kHz carrier with a sinc pulse. The received signal is then cross-correlated with the same sinc pulse to find the first peak above a certain threshold. In order to speed up the cross correlation, it is performed using overlapping Fast Fourier Transforms (FFTs) [18]. Figure 5 shows the recorded ultrasound pulse and the cross correlation result from an anchor that is placed 10 m away. It can be seen that the noise amplitude is rejected on the correlated signal, making analysis of the distance easier.

V. Position Estimation

The software on the quadcopter mainboard does the bulk of all the computational work required for the copter to operate. This section gives a brief overview of the discrete-time calculation performed in software to update the state of the system at time n regularly with interval dt.

The algorithm that runs at the highest rate of the control system is the attitude estimation and control. We have used a slightly modified version of an Attitude and Heading Reference System (AHRS) algorithm [19] to get a quaternion-based representation of the current attitude, from which we calculate Euler angles as:

$$\begin{bmatrix}\theta_r(n)\\ \theta_p(n)\\ \theta_y(n)\end{bmatrix} = \begin{bmatrix}atan2(2(q_0q_1+q_2q_3), 1-2(q_1^2+q_2^2))\\ arcsin(2(q_0q_2-q_3q_1))\\ atan2(2(q_0q_3+q_1q_2), 1-2(q_2^2+q_3^2))\end{bmatrix} \quad (1)$$

where $[q_0, q_1, q_2, q_3]^T$ is the quaternion representation of the current attitude, $atan2$ is a function

for $arctan$ that takes two arguments to handle all possible angles and $[\theta_r(n), \theta_p(n), \theta_y(n)]^T$ are the roll, pitch, and yaw Euler angles. Then, there is one Proportional-Integral-Derivative (PID) controller for each Euler angle to stabilize the copter. There is also a PID controller for the altitude. In order to get as little altitude-variations as possible, feed-forward is used on the throttle output from the roll and pitch-angles, calculated as:

$$FF_{fac}(n) = \sqrt{tan(\theta_r(n))^2 + tan(\theta_p(n))^2 + 1} \quad (2)$$

The feed-forward term $FF_{fac}(n)$ is calculated at a higher rate than the altitude measurements arrive and represents a compensation factor that makes the vertical thrust component constant while the roll and pitch angles $[\theta_r(n), \theta_p(n)]^T$ vary. This equation has singularities when the roll or pitch angles are at $90°$, but the attitude control loop truncates its inputs to prevent the roll and pitch angles from exceeding $45°$.

After several manual aggressive flight tests with altitude hold activated, we had confidence that our altitude control loop was working properly.

To estimate the position of the copter, we use one high-rate update based on dead reckoning from its attitude. The assumption is that the throttle is controlled such that the altitude remains constant or slowly changing. Additionally, one low-rate update is used on the position every time new ultrasound ranging values arrive. For the high-rate dead reckoning, the first thing we calculate for each iteration is the velocity-difference $[d_{vx}(n), d_{vy}(n)]^T$ and add it to the integrated velocity value $[V_x(n), V_y(n)]^T$, rotated by the yaw angle:

$$\begin{bmatrix} d_{vx}(n) \\ d_{vy}(n) \end{bmatrix} = \begin{bmatrix} 9.82tan(\theta_r(n) + \theta_{rofs}(n))dt \\ 9.82tan(\theta_p(n) + \theta_{pofs}(n))dt \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} c_y \\ s_y \end{bmatrix} = \begin{bmatrix} cos(\theta_y(n)) \\ sin(\theta_y(n)) \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} V_x(n) \\ V_y(n) \end{bmatrix} = \begin{bmatrix} V_x(n-1) + d_{vx}(n)c_y + d_{vy}(n)s_y \\ V_y(n-1) + d_{vx}(n)s_y + d_{vy}(n)c_y \end{bmatrix} \quad (5)$$

where $\theta_{rofs}(n)$ and $\theta_{pofs}(n)$ are offsets that could be estimated over time to compensate for misalignment of the accelerometer. Again, the singularity when the roll or pitch angle $[\theta_r(n), \theta_p(n)]^T$ are $90°$ is not an issue because these angles are limited at $45°$. This is then used to update the position $[P_x(n), P_y(n)]^T$:

$$\begin{bmatrix} P_x(n) \\ P_y(n) \end{bmatrix} = \begin{bmatrix} P_x(n-1) + V_x dt \\ P_y(n-1) + V_y dt \end{bmatrix} \quad (6)$$

As the velocity integration drift is unbounded even when there is a small offset on the attitude estimation, the anchor distance measurements have to be used to estimate the velocity drift in addition to the roll and pitch error. For the anchor corrections, which arrive at a lower rate, we first compute the difference between the expected distance to the anchor from the dead-reckoning and the measured distance to the anchor:

$$\begin{bmatrix} d_{ax}(n) \\ d_{ay}(n) \\ d_{az}(n) \end{bmatrix} = \begin{bmatrix} P_x(n-1) - P_{x,anchor} \\ P_y(n-1) - P_{y,anchor} \\ P_z(n-1) - P_{z,anchor} \end{bmatrix} \quad (7)$$

$$d_a(n) = \sqrt{d_{ax}(n)^2 + d_{ay}(n)^2 + d_{az}(n)^2} \quad (8)$$

$$err(n) = d_a(n) - d_{measured}(n) \quad (9)$$

$$F_c(n) = \frac{err(n)}{d_a(n)} \quad (10)$$

where $[P_x(n), P_y(n), P_z(n)]^T$ is the position of the copter, $[P_{x,anchor}, P_{y,anchor}, P_{z,anchor}]^T$ is the position of the anchor this measurement came from and $[d_{ax}(n), d_{ay}(n), d_{az}(n)]^T$ is the difference between them. Further, $d_a(n)$ is the magnitude of the calculated difference, $d_{measured}(n)$ is the measured magnitude, $err(n)$ is the difference between the calculated and the measured magnitude and $F_c(n)$ is a factor that is used in later calculations for correction. Notice that there is a singularity when $d_a(n)$ approaches 0, but this would imply that the copter is located exactly on one anchor which means that the copter collides with that anchor. This should not happen because the copters should keep a safety distance from the anchors at all times.

At this point, if the error is larger than a certain threshold, we discard this measurement and lower the position quality because something is likely to be wrong. If too many consecutive measurements have a large error, we stop discarding and start using them in case this is the initial position correction at start-up.

Next, the position differences $[d_{ax}(n), d_{ay}(n), d_{az}(n)]^T$ are used to correct the current position and the velocity error where we compute proportional and derivative parts, $[P_{xpos}(n), P_{ypos}(n)]^T$ and $[D_{xpos}(n), D_{ypos}(n)]^T$, on the position error. The gain components in the following equations $(G_{p,vel}, G_{p,pos}, G_{d,pos})$ were derived experimentally and the simulation presented in Section VII has been an important aid for doing that.

$$\begin{bmatrix} P_{xpos}(n) \\ P_{ypos}(n) \end{bmatrix} = \begin{bmatrix} d_{ax}(n)F_c G_{p,pos} \\ d_{ay}(n)F_c G_{p,pos} \end{bmatrix} \quad (11)$$

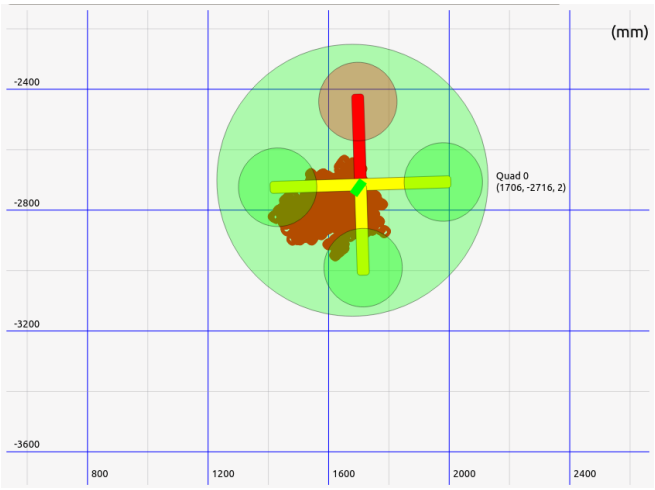

Fig. 6. The estimated position during a 60 s long hover.

$$\begin{bmatrix} D_{xpos}(n) \\ D_{ypos}(n) \end{bmatrix} = \begin{bmatrix} (d_{ax}(n)F_c - d_{ax}(n-1)F_c)G_{d,pos} \\ (d_{ay}(n)F_c - d_{ay}(n-1)F_c)G_{d,pos} \end{bmatrix} \quad (12)$$

Then, apply this to the position:

$$\begin{bmatrix} P_x(n) \\ P_y(n) \end{bmatrix} = \begin{bmatrix} P_x(n-1) + P_{xpos}(n) + D_{xpos}(n) \\ P_y(n-1) + P_{ypos}(n) + D_{ypos}(n) \end{bmatrix} \quad (13)$$

The height $P_z(n)$ could also be updated as above, but using the ultrasound sensor directed towards the floor directly gave better results in our experiments.

Updating the velocity state $[V_x(n), V_y(n)]^T$ is done in a similar way:

$$\begin{bmatrix} V_x(n) \\ V_y(n) \end{bmatrix} = \begin{bmatrix} V_x(n-1) + d_{ax}F_c(n)G_{p,vel} \\ V_y(n-1) + d_{ay}F_c(n)G_{p,vel} \end{bmatrix} \quad (14)$$

A test flight of 60 s where a simple PID control loop is issuing control commands to hold the $[x, y]^T$ position based on the estimated position is shown in Figure 6. The overlapping red dots represent estimated position samples during this flight, and it can be seen that the deviation was below 20 cm for the entire flight. Notice that we did not have a more accurate positioning system to compare with in this test. The distribution of the estimated position gives an impression about the performance.

Because of the complexity we did not attempt to make an analytical stability analysis of the position-estimation algorithm. We did an experimental stability analysis using fault injection presented in Section VII-A.

VI. Collision Avoidance

In this study, collision avoidance is attempted by placing risk contours around copters and static objects from the perspective of every copter, and steering away if the risk contours overlap with the comfort zone of the copter [20]. This means that the risk contours are not a global state, but different from every copter's perspective based on its relative velocity to the object and when the positions of other copters were last received. The comfort zone is represented as a circle placed around the quadcopter with a radius that is calculated based on the confidence of the position estimation.

The risk contours are represented as ellipses and sized/rotated based on the squared relative velocity vector to the copters/objects they surround. To share the knowledge about the position of all copters, they broadcast this information one at a time to everyone else. When a copter receives a position update from another copter, it will update its Local Dynamic Map (LDM) with this information. Between the position updates, the risk contours around other copters will be moved and reshaped based on the velocity the other copters had when their position was last received. What this looks like can be seen in the screenshot in Figure 7 of the visualization and control program we developed for this application.

The risk contour around every neighbouring object in each copter's LDM looks like the following:

$$\begin{bmatrix} d_{vx} \\ d_{vy} \end{bmatrix} = \begin{bmatrix} V_{x,r} - V_x \\ V_{y,r} - V_y \end{bmatrix} \quad (15)$$

$$d_v = \sqrt{d_{vx}^2 + d_{vy}^2} \quad (16)$$

where $[d_{vx}, d_{vy}]^T$ are the X and Y velocity difference between this copter's own velocity and the velocity $[V_{x,r}, V_{y,r}]^T$ of the copter corresponding to this risk contour in the LDM. The position $[R_{px}, R_{py}]^T$, width, height $[R_w, R_h]^T$ and rotation θ_r of the risk contour are calculated as:

$$\begin{bmatrix} R_{px} \\ R_{py} \end{bmatrix} = \begin{bmatrix} P_{x,c} + R_{gx}d_{vx}d_v \\ P_{y,c} + R_{gx}d_{vy}d_v \end{bmatrix} \quad (17)$$

$$\begin{bmatrix} R_w \\ R_h \end{bmatrix} = \begin{bmatrix} R_r + R_{gx}d_{vx}d_v^2 \\ R_r + R_{gy}d_{vy}d_v^2 \end{bmatrix} \quad (18)$$

$$\theta_r = atan2(d_{vy}, d_{vx}) \quad (19)$$

where R_r is a safety margin around the copter in the LDM that this risk contour surrounds. R_r is scaled based on the time that has passed since the copter corresponding to this risk contour was heard the last time. $[P_{x,c}, P_{y,c}]^T$ is the position of the copter corresponding to this risk contour. Further,

R_{gx} and R_{gy} are factors that scale the size of the risk contour that we found suitable values for in the auto-generated tests described in Section VII.

When an overlap between the comfort zone of a copter and a risk contour occurs, the collision-avoidance mechanism will take over control and steer away from the overlapping risk contour in the opposing direction. If there are several simultaneous overlaps, a vector will be calculated from a weighted sum of all overlapping risk contours and their relative direction, and used to steer away from the collision, calculated as:

$$\begin{bmatrix} C_x \\ C_y \end{bmatrix} = \sum_{i=0}^{N} \begin{bmatrix} C_{x,i} M_i \\ C_{y,i} M_i \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} C_r \\ C_p \end{bmatrix} = \begin{bmatrix} -cos(\theta_y) C_x - sin(\theta_y) C_y \\ -sin(\theta_y) C_x + cos(\theta_y) C_y \end{bmatrix} \tag{21}$$

where $[C_x, C_y]^T$ are the relative $[X, Y]^T$ direction sums of all risk contours i that overlap with the comfort zone of the copter. $[C_r, C_p]^T$ are the roll and pitch output commands calculated from all overlapping risk contours, rotated by the yaw angle θ_y of the copter. Further, M_i is the amount of overlap with every overlapping risk contour i. Thus, the more overlap there is for one risk contour, the more influence it will have on the output.

It should be noted that collision avoidance is done in two dimensions. This is because our copters are not able to fly over each other even if they are at different heights, since the height sensor of each copter requires a free path to the ground. Since the position-estimation algorithm relies on an altitude controller that keeps the altitude constant or slowly changing, collision avoidance in the Z direction is not necessary if truncation is used on the set point of the altitude controller.

VII. Simulation and Fault Injection

To evaluate and optimize our quadcopter system, we have created a simulator with the architecture shown in Figure 8. Our simulator is a library written in C++ with an interface where copters can be added, removed, or commanded to move. The block named *coptermodel* runs the same code for position and velocity estimation, shown in Equation 5 and 6, as the real implementation on the hardware copters. The angles $[\theta_r, \theta_p, \theta_y]^T$ are updated from the movement command with a similar response to that of the actual hardware, and then the position and velocity state is updated based on these angles. For this update, we do not inject any faults and assume that it represents the true position of the copter.

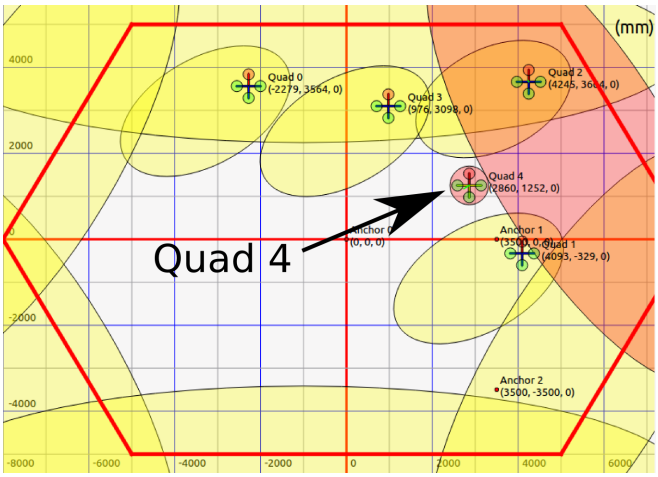

Fig. 7. A screenshot of the risk contours from the perspective of Quad 4. The red contour is red because there is an overlap between the copters own comfort zone (Quad 4) and the risk contour around the right upper wall.

There is also a position and velocity state that is updated in the same way, but where we inject various faults. This perceived position is then corrected from simulated ultrasound measurements as described in Equation 13 and 14, while we inject faults on these ultrasound measurements. Additionally, each simulated copter has the collision-avoidance mechanism described in Section VI, shown as Intelligent Transportation System (ITS)-station in Figure 8. The simulated ITS-station on each copter broadcasts and receives ITS-messages to and from the other copters every 100 ms, where we also inject faults. The CopterSim library can either be used from a Graphical User Interface (GUI) to manually add and move copters, or from a program that auto-generates tests and injects faults. All fault injection is done with probes from the FaultCheck tool [10], linked to the simulator.

We have created a model for the QuickCheck tool [11] that sends commands to the simulator where we add a random number of copters at random non-overlapping positions and run commands while checking the property that they do not collide. These randomly-generated commands can either be steering commands for the copters, or fault-

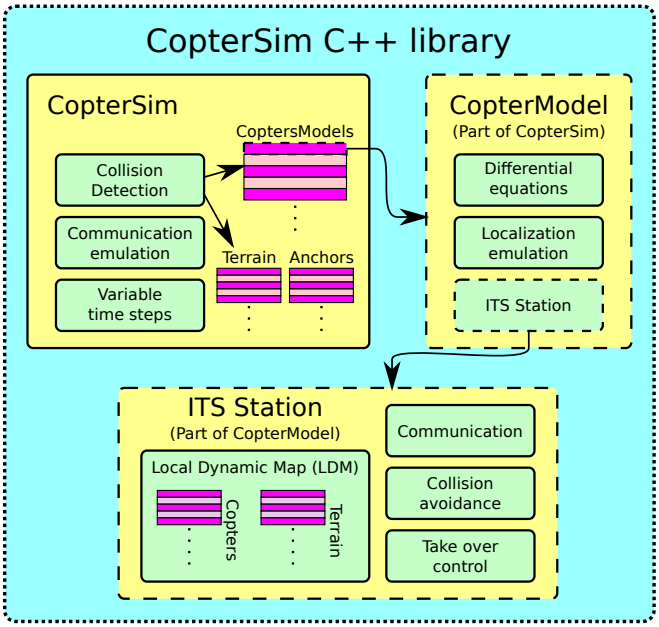

Fig. 8. The simulation architecture. CopterSim has a list with all copters and checks for collisions between them and the terrain in every iteration. CopterModel handles the physical model of every copter and has an ITS-station that handles collision avoidance. The localization emulation is also part of CopterModel.

injection commands passed to the FaultCheck tool. The whole set-up can be seen in Figure 9.

The parameters for the steering commands are:

- Which copter to command, randomly chosen from all the copters present in the simulation.
- The roll output, randomly chosen between $\pm 15°$.
- The pitch output, randomly chosen between $\pm 15°$.
- The yaw rate output, randomly chosen between $\pm 90°$ per second.

Further, the fault injection commands have the following parameters:

- Which copter to affect, randomly chosen from all the copters present in the simulation.
- The fault type, randomly chosen from:
  - Communication bit-flip, which flips a randomly chosen bit of the broadcast ITS message.
  - Packet loss, where ITS-messages are lost.
  - Repetition, where ITS-messages are repeated.
  - Ultrasound ranging faults, where a random offset is added to the ultrasound distance measurements.

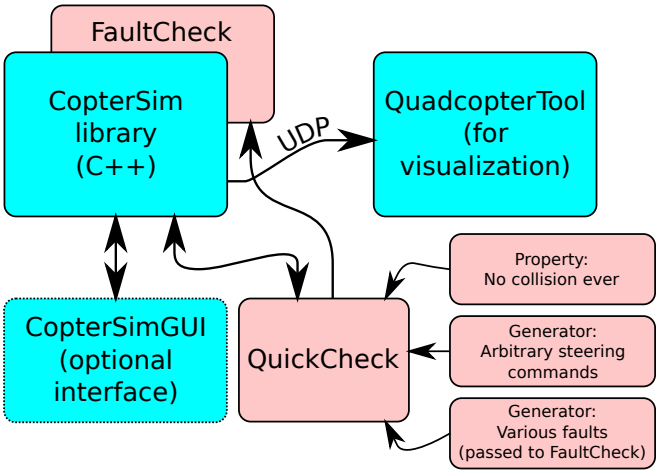

Fig. 9. CopterSim with FaultCheck connected to a visualization tool and QuickCheck.

  - Offsets on the $[\theta_r, \theta_p, \theta_y]^T$ angles.

When a collision occurred during these auto-generated tests, we used the QuickCheck tool to shrink the sequence of commands to a smaller one that led to a collision, in order to make it easier to identify the problem. Then, we replayed this smaller sequence of commands while adjusting the gains described in Section V and the risk contour parameters described in Section VI until this command sequence did not lead to a collision any more. This also gave us insight into the number of simultaneous faults the copters can handle.

A. Experimental Performance and Stability Analysis

Here we show specific injected faults and their impact on the position and velocity error under different gain values, which are described in Section V. This is not a full analysis of all possible combinations of faults and gains, but it gives a general impression about the fault tolerance and performance of the system under different conditions. Running many auto-generated tests with different combinations of injected faults gave us confidence that the chosen parameters gave a robust position correction.

Figure 10 shows how the position recovers when a position offset fault of 1.5 m is injected. The left part of the graph shows the recovery with $G_{p,pos} = 0.2$ and the right part with $G_{p,pos} = 1.0$. It can be seen that the higher position gain makes the position error recover faster. A similar relation is

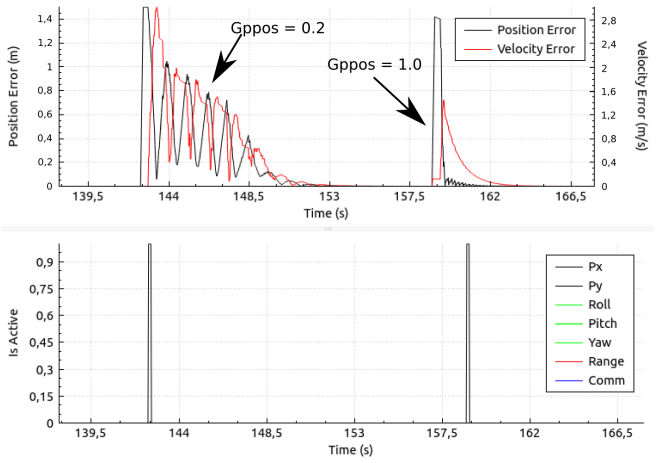

Fig. 10. Fault injection with 1.5 m position offset. The lower part shows when the faults are active and the upper part shows the position and velocity errors. The first activation had $G_{p,pos} = 0.2$ and the second one had $G_{p,pos} = 1.0$.

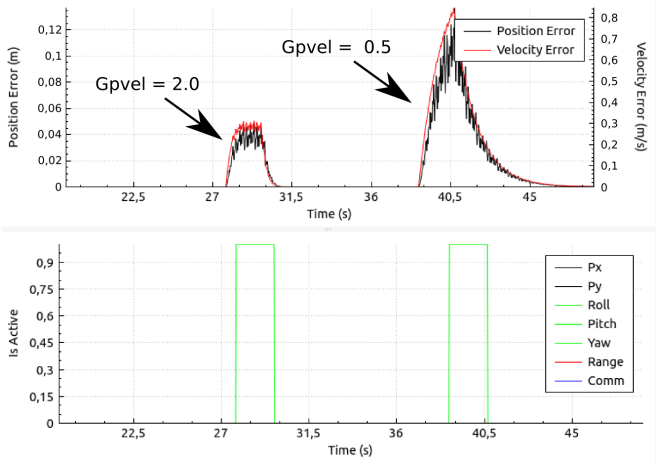

Fig. 11. Fault injection with 5° pitch offset. The lower part shows when the faults are active and the upper part shows the position and velocity errors. The first activation had $G_{p,vel} = 2.0$ and the second one had $G_{p,vel} = 0.5$.

shown in Figure 11 where a pitch offset is injected for different values of $G_{p,vel}$. When the gain is too high, an oscillation such as in Figure 12 where a position offset of 1.5 m is injected while $G_{p,pos} = 2.0$ can occur.

In Figure 13 a position offset fault is shown with $G_{p,vel} = 0.0$ and $G_{p,vel} = 2.0$. It can be seen that when only a position offset is injected, the velocity gain does not help at all. However, a roll or pitch offset such as in Figure 11 requires $G_{p,vel} > 0$ to recover. An example where both a ranging offset and a pitch offset are injected at the same time can be seen in Figure 14, where the difference between low and high $G_{p,vel}$ can be seen. The injected pitch fault requires $G_{p,vel} > 0$, but the ranging fault recovers the same way with lower $G_{p,vel}$.

A dynamic example, where one copter is moved forth and back, is shown in Figure 15 for different values of $G_{p,pos}$ while an amplification on the pitch of 0.95 is injected. If there were not any acceleration the pitch amplification fault would remain unnoticed, but the acceleration makes it appear. It can be seen that higher gain keeps the position error lower during the flight.

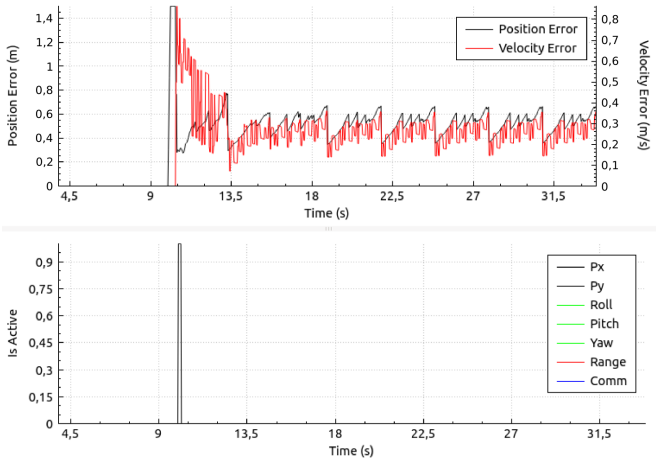

Fig. 12. Fault injection with 1.5 m position offset and $G_{p,pos} = 2.0$. The lower part shows when the faults are active and the upper part shows the position and velocity errors.

VIII. Conclusions

We have created a quadcopter platform that has a novel approach to localization using ultrasound distance measurement combined with IMU-based dead reckoning for accurate positioning, while we use risk contours to avoid collisions with static objects and other copters. Additionally, we have created a powerful simulation environment where we can auto-generate tests and inject faults with many copters simultaneously, making it possible to scale up the tests beyond what our hardware allows. Our current platform has a limited size, because the anchors can be no further away from the copters than 12 m. Future work includes implementation of handover, both in simulation and hardware, between flying zones to handle more anchors spread out in a larger area. Another improvement would be handover between GNSS positioning and the

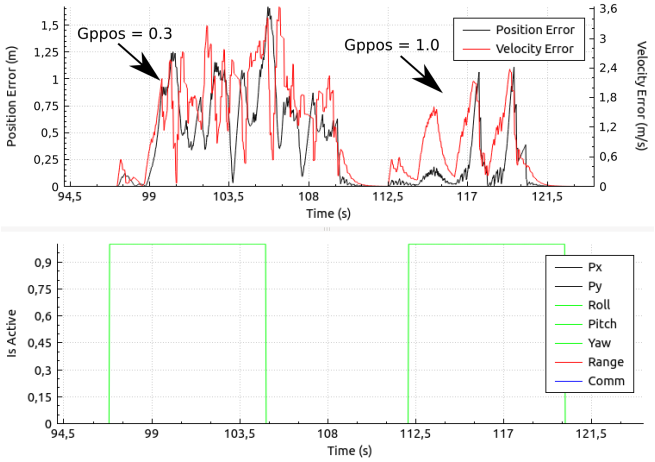

Fig. 15. Fault injection with 0.95 pitch amplification. The lower part shows when the faults are active and the upper part shows the position and velocity errors. The first activation had $G_{p,pos} = 0.3$ and the second one had $G_{p,pos} = 1.0$.

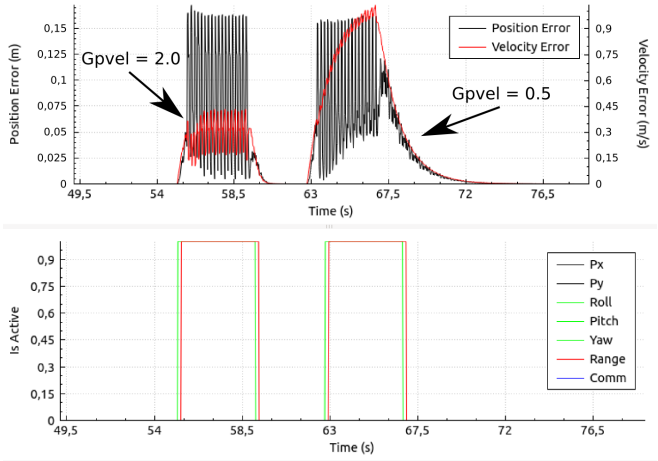

Fig. 14. Fault injection with 5° pitch offset and 0.5 m anchor distance offset. The lower part shows when the faults are active and the upper part shows the position and velocity errors. The first activation had $G_{p,vel} = 2.0$ and the second one had $G_{p,vel} = 0.3$.

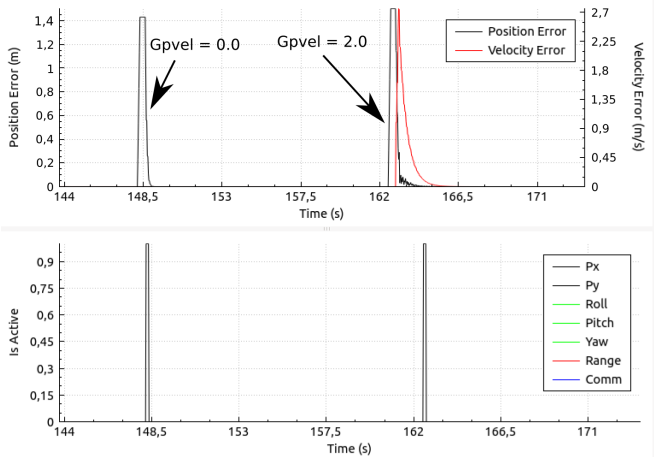

Fig. 13. Fault injection with 1.5 m position offset. The lower part shows when the faults are active and the upper part shows the position and velocity errors. The first activation had $G_{p,vel} = 0.0$ and the second one had $G_{p,vel} = 2.0$.

positioning method proposed in this work, when higher position accuracy is required during landing and take-off.

IX. Acknowledgement

This research has been funded through the KARYON EU project (Grant agreement no: 288195), the PROWESS EU project (Grant agreement no: 317820) and through EISIGS (grants from the Knowledge Foundation). We would like to thank Kenneth Östberg for fruitful discussions regarding the usage of risk contours.

References

[1] M. Achtelik, T. Zhang, K. Kuhnlenz, and M. Buss, "Visual Tracking and Control of a Quadcopter Using a Stereo Camera System and Inertial Sensors," in *Proceedings of the International Conference on Mechatronics and Automation (ICMA)*, 2009, pp. 2863–2869.

[2] B. Ben Moshe, N. Shvalb, J. Baadani, I Nagar, and H. Levy, "Indoor Positioning and Navigation for Micro UAV Drones," in *Proceedings of the 27th Convention of Electrical Electronics Engineers in Israel (IEEEI)*, 2012, pp. 1–5.

[3] M Bošnak, D Matko, and S Blažič, "Quadrocopter Hovering Using Position-Estimation Information from Inertial Sensors and a High-delay Video System," *Journal of Intelligent & Robotic Systems*, vol. 67, no. 1, pp. 43–60, 2012.

[4] J. Engel, J. Sturm, and D. Cremers, "Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing," in *Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, 2012.

[5] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a Navigation System for Autonomous Indoor Flying," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 2878–2883.

[6] I. Sa and P. Corke, "System Identification, Estimation and Control for a Cost Effective Open-Source Quadcopter," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 2202–2209.

[7] B. Vedder. (2014). CC2520 and STM32F4 RF Boards, [Online]. Available: `http://vedder.se/2013/04/cc2520-and-stm32-rf-boards/`.

[8] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, 1990.

[9] J. Derrick, N. Walkinshaw, T. Arts, C. Benac Earle, F. Cesarini, L. Fredlund, V. Gulias, J. Hughes, and S. Thompson, "Property-Based Testing - The ProTest Project," in *Formal Methods for Components and Objects*, ser. Lecture Notes in Computer Science, F. Boer, M. Bonsangue, S. Hallerstede, and M. Leuschel, Eds., vol. 6286, Springer Berlin Heidelberg, 2010, pp. 250–271.

[10] B. Vedder, T. Arts, J. Vinter, and M. Jonsson, "Combining Fault-Injection with Property-Based Testing," in *Proceedings of the International Workshop on Engineering Simulations for Cyber-Physical Systems*, ser. ES4CPS '14, Dresden, Germany: ACM, 2014, 1:1–1:8.

[11] T. Arts, J. Hughes, J. Johansson, and U. Wiger, "Testing Telecoms Software with Quviq QuickCheck," in *Proceedings of the ACM SIGPLAN Workshop on Erlang*, Portland, Oregon: ACM Press, 2006.

[12] G. Hoffmann, D. Rajnarayan, S. Waslander, D. Dostal, J. S. Jang, and C. Tomlin, "The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC)," in *The 23rd Digital Avionics Systems Conference, DASC 04*, vol. 2, 2004, 12.E.4–121–10 Vol.2.

[13] J. Pestana, J. Sanchez-Lopez, P. de la Puente, A Carrio, and P. Campoy, "A Vision-Based Quadrotor Swarm for the Participation in the 2013 International Micro Air Vehicle Competition," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 617–622.

[14] J. Eckert, R. German, and F. Dressler, "On Autonomous Indoor Flights: High-Quality Real-Time Localization Using Low-Cost," in *IEEE International Conference on Communications (ICC 2012), IEEE Workshop on Wireless Sensor Actor and Actuator Networks (WiSAAN 2012)*, Ottawa, Canada: IEEE, 2012, pp. 7093–7098.

[15] J. F. Roberts, T. Stirling, J. Zufferey, and D. Floreano, "Quadrotor Using Minimal Sensing for Autonomous Indoor Flight," in *European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, Toulouse, France, 2007.

[16] J. Eckert, R. German, and F. Dressler, "ALF: An Autonomous Localization Framework for Self-Localization in Indoor Environments," in *7th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2011)*, Barcelona, Spain: IEEE, 2011, pp. 1–8.

[17] J. Eckert, *Autonomous Localization Framework for Sensor and Actor Networks: Autonomes Lokalisierungsframework Für Sensor- und Aktornetzwerke*. 2012.

[18] J. Jan, *Digital Signal Filtering, Analysis and Restoration*, ser. IEE telecommunications series. Institution of Electrical Engineers, 2000.

[19] S. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm," in *IEEE International Conference on Rehabilitation Robotics (ICORR)*, 2011, pp. 1–7.

[20] K. Östberg et al., "Safety Constraints and Safety Predicates," KARYON consortium, Public Report, 2014.