Systems Article

Modular Design and Implementation for Rapid Deployment of Autonomous Systems

Tobias Stenbock Andersen[1], Ananda Narasipur Holck Nielsen[2], Matteo Fumagalli[2], Joachim Langtved Axelsen[2], Mads Christiansen[2], Ole Ravn[2] and Nils Axel Andersen[2]

[1]Department of Electrical Engineering, Technical University of Denmark (DTU), Elektrovej, Building 326, 2800 Kgs. Lyngby, Denmark

[2]DTU, Kgs. Lyngby, Denmark

Abstract: This paper describes the autonomous systems, which Technical University of Denmark used to participate in Challenge 2 of the Mohamed Bin Zayed International Robotics Challenge. We participated with two autonomous vehicles in the Challenge: an aerial and a ground vehicle. The mission of both of the vehicles was to locate blocks and use them to build a wall in a marked location. Our ground solution consisted of a SKID steered vehicle, with a Universal Robots arm attached to it, and our aerial solution was a DJI M100 quadrotor in X configuration, equipped with a rangefinder and camera. Both platforms each have their own custom build end-effector, designed for lifting flat magnetic objects. The software was designed with a modular approach based on the mobotware framework, such that mission scripts could rapidly be assembled at the deployment site. A state of the art neural network, for detecting blocks, was trained for our ground vehicle. The effectiveness of the modular approach was tested in the challenge, and our lessons learned is included in the paper.

Keywords: modularity, autonomous robot, robot perception, learning, manipulators, MBZIRC

1. Introduction

When designing and implementing complex systems it is often found, that the system requirements change significantly during the project period, and even during deployment, the real operation conditions may require substantial changes to the solution. This is especially the case, when it is difficult to foresee these conditions, e.g. disaster response and emergency systems.

An example of a system with highly dynamic system requirements is the Mohammed Bin Zayed International Robotics Challenge (MBZIRC)). It is a large international robotics competition held biennially in Abu Dhabi, with participation of elite universities from all over the world. It consists of three challenges and a grand challenge that is a combination of the three challenges at the same time. The challenges favours autonomous solutions and allows for both Unmanned Ground Vehicles (UGVs) and Unmanned Aerial Vehicles (UAVs) to participate. A qualitative description of the

Received: 30 September 2020; revised: 30 June 2021; accepted: 7 August 2022; published: 1 November 2022.

Correspondence: Nils Axel Andersen, DTU, Kgs. Lyngby, Denmark, Email: naa@elektro.dtu.dk

DOI: https://doi.org/10.55417/fr.2022063

challenges with a video visualisation is published approximately two years before the competition, then the description is refined several times after the chosen teams have handed in two progress reports. The final description comes approximately 3 month before the competition, and when you arrive you find that some of the real conditions differ so much from the final description, that a change of solution is needed. For each challenge three time slots of 15 minutes are given to test the solutions in the real arena before the actual competition, meaning the deployment time on the actual deployment site is limited. While the MBZIRC challenges are less cluttered and better well defined than real life challenges, they still contain a lot of the same problems in the form of the unexpected changes and time limited deployment, and a lot can be learned from solving these challenges, although they are a simpler version of real world problems.

It is clear that using the classic waterfall model (Royce 1970) to design a monolithic solution to such a problem is likely to fail, as it will be difficult to adapt the system to the changes. A better way will be to use the spiral model (Boehm 1988) and a modular solution. Our contribution is the implementation of a modular software approach on our platforms, as preparation for the MBZIRC. The modular solutions have been tested in our local lab environment, before they were taken to Abu Dhabi and assembled into mission scripts. Our platforms participated in the MBZIRC, thereby allowing for a real life test of the usability of a modular approach, which provides valuable lessons. This paper will describe our solution to MBZIRC 2020 with emphasis on challenge 2, as well as, the lessons learned from using a modular approach in such an environment.

2. Challenge overview

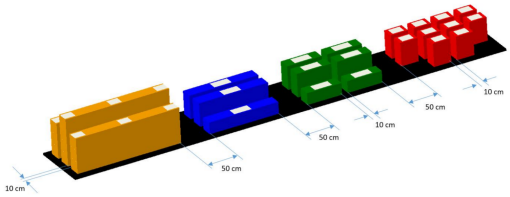
In all the MBZIRC challenges, autonomous solutions was favoured to a point where a fully autonomous solution, which scored any amount of points, always would rank higher than any teleoperated solution. In Challenge 2 of MBZIRC 2020, up to three UAVs and a single UGV was allowed. The two Unmanned vehicles (UVs) had the same tasks in this challenge, and the tasks could be done in parallel. The UVs had to locate and lift blocks of varying size and colour. There were four block types in total: red, green, blue and orange. The blocks had varying sizes as seen on Figure 1. The blocks rewarded points accordingly to their size, meaning that the orange block gave the most amount of points, and the red the least amount of point, furthermore, the UAVs received more points than the UGV, when placing the same coloured blocks. These blocks had to be transported to a marked but unknown wall position, and placed in a specific foreknown pattern. The blocks had metal plates attached on the top, meaning magnets could be used to lift the blocks.

The two platforms each had their own stacks, which was designed to make it easier for the associated platform to locate and lift the blocks. The UGV stacks were compact groups of same coloured blocks placed next to each other on a line, and the UAV stacks were lines of individual block. The blocks, of the UAV, were surrounded by walls such that they would not blow away when approached by the UAV. Figure 1 shows the setup of the two types of stacks. Exact details on the competition can be found on the MBZIRC website[1]. The latest iteration of the challenge is found in the documents linked in bottom of the challenge page.

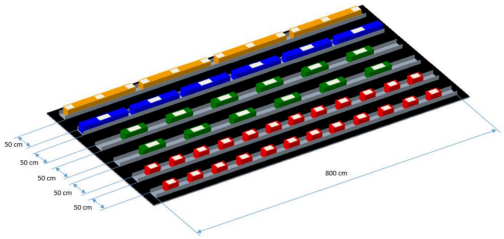
Experience showed that the position of the stacks was constant for each competition day. Meaning that once the competition day started, and the stacks had been placed, they would not change position until the next day.

Similarly to the stack designs, the walls were also made to fit the strengths and weaknesses of the different platforms. The UGV had to build the wall from the ground up and the UAVs had to build the wall high in the air. To mark the position of where the UGV had to build the wall, an L formed yellow-magenta checkers pattern was laid out on the ground. This L formed marker will be denoted the "L" in the rest of the paper. The UAVs had a zigzag shaped wall which was 1.7 meters tall, and the UAVs had to place their blocks on top of the wall. Depictions of the two types of walls

[1] https://web.archive.org/web/20210618232441/https://www.mbzirc.com/challenge/2020

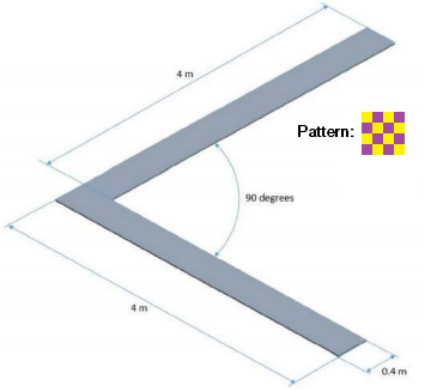


(a) The UGV stacks, where it can be seen that the same coloured blocks are stored in groups.

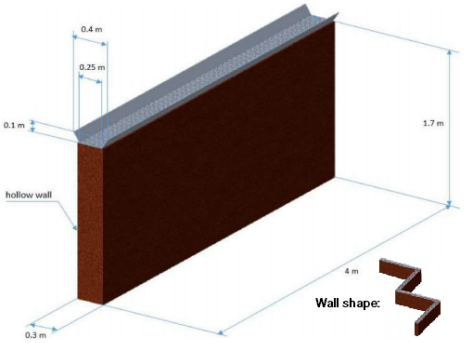


(b) The UAV stacks, which was placed in a U shaped container to prevent the blocks from being blown away by the UAV.

Figure 1. The two types of stack groups, which the UV's had to collect the blocks from. (The graphics are from the Challenge descriptions, which the Khalifa university sent out to the participants.)



(a) The UGV L formed wall marking. The L wall had a yellow-magenta checkers pattern, seen in the figure.



(b) The UAV zizag wall. The UAV wall contained a U-shaped wire mesh, which would help guide the blocks.

Figure 2. The two types of walls where the UVs should place the blocks. (The graphics are from the Challenge descriptions, which the Khalifa university sent out to the participants.)

can be seen on Figure 2. The UGV would always have to build one completely orange wall on one side of the "L" and a wall consisting of one blue, two green and four red blocks on the other side of the "L". The multicoloured wall would have a random pattern. The UAVs would always have an orange wall, and a random pattern of blue, green and red blocks, on the remaining three walls. The patterns would be given at the start of each run, and not following the them would penalize the score. Additional information on the score can be found on the MBZIRC website[2].

The above description was of the finalised challenge, but the challenge had other iterations[3] during the design process of the solution, which resulted in an additional complication as mentioned in Section 1. The solutions created, therefore, had to be modular, such that they could easily be changed or fitted to accommodate any potential changes to the challenge. Much of the final challenge was the same as the early challenge, but there were two major differences. The first one being, that the early challenge description shows the UAVs and UGV collecting blocks from the same area and building the same wall, instead of two separate walls, meaning that the early challenge did not have the same parallel approach. The second one is, that the blocks in the early challenge are initially placed chaotically, and not neatly stacked to ease the lifting challenge of the platforms.

[2] https://web.archive.org/web/20210410150534/https://www.mbzirc.com/scoring-scheme
[3] https://youtu.be/l5aPjTNYcpc?t=60

2.1. UGV tasks

The UGV had to be able to fulfil several tasks to complete the challenge. The first task was detecting the blocks, which can be done with visual or laser feedback.

The second task was lifting the blocks. This requires the UGV to have hardware, that allows it to manipulate the blocks. Electromagnets, vacuum lifting and gripper mechanisms are all valid options.

There were two navigation tasks for the UGV. The first one is navigating to the correct stacks. Due to the object sparse environment of the arena, algorithms such as SLAM might not be very reliable, and opting for custom built point based navigation, which relies on pre-obtained knowledge could be more robust. The second navigation task was to locate and navigate to the "L". This task is twofold, as when locating the "L" before any blocks have been placed, only visual feedback is possible, but once the first layer have been placed, the same visual feedback, might not be usable, as the "L" would be partly obscured, and a second "L" detection method is likely needed.

The next task was the wall building. The UGV has to be able to place the blocks close enough, that the entire wall remains within the wall marker and that the wall remains stable. All this must be done without miss-placing the blocks such that they bump into each other, since this creates a risk of dropping the block or overturning the wall.

Driving back and forth between the stacks and the wall is be a time consuming task. To alleviate this the UGV should be able to move as many blocks as possible at once. This results in a need for a storage space on top of the UGV.

2.2. UAV tasks

As with the UGV's challenge, the UAV's challenge can also be divided into smaller tasks, which need to work in order for the challenge to be completed.

The first task was navigation, as the UAV must be able to move around the arena, and be able to move to the points of interest. This navigation module is needed firstly to search for the blocks, but also to move between the stacks and the wall afterwards. The next task was detection of the blocks as it is necessary to be able to analyse the sensory data to identify a suitable block in the environment. This module should also be able to provide an estimate of the position of the identified block to be used as a feedback for the third task - the pickup. The pickup is both a hardware, as well as, software task, as a mechanical contraption of some kind is necessary for interacting with the block and thus the environment. It is important that the pickup mechanism is compliant enough to not make the drone unstable during the pickup.

2.3. Strategy

Our strategy for winning MBZIRC was to have the UGV build 2-3 full layers of the coloured wall and 2-3 half layers of the orange wall. The UGV would do this by fully packing one coloured layer and one half orange layer, and placing the entire layer in one go. This would reduce the amount of time needed to drive back and forth between the wall and the stacks, and optimise the time spent building the wall. It was estimated that 2-3 layers was the maximum amount of layers, which the UGV could achieve within the time of the challenge. While the UGV was building the wall, the UAV was to collect as many red blocks as it could and place them on the wall. It was estimated that placing the red blocks with the UAV had the highest chance of success. The UAV was to ignore the colour ordering of the wall, to simplify the challenge, and simply aim to gain fractional points.

An earlier strategy, which affected some design decisions, involved the cooperation of the UAV and the UGV. This strategy was made when the earlier challenge description was the only one available, and the UVs had to build a common wall. In that setup, placing blocks on the wall is challenging for the UAV for two reasons: the first reason is that the blocks are light and the UAV will blow the wall away when trying to place the subsequent blocks. The second reason is that placing

the blocks precisely is difficult for the UAV due to its aerial nature. Compared to the UAV, this task is easy for the UGV. It was, therefore, decided to have the UAV collect blocks and drop them off near the wall, where the UGV could pick them up and place them correctly on the wall. This would also support the UGV, as it uses a large amount of time to drive around and collect the necessary blocks. With the UAV doing the block collecting, the UGV could focus on building the wall.

Early testings of this strategy proved very successful, and further development was, therefore, made with this strategy in mind, until the finalised challenge description was revealed. Due to the parallel nature of the final challenge, the cooperation strategy no longer made sense and strategy was discarded.

3. Modularity

The general idea of modularisation of a complex system is to break it into smaller parts that are simpler to design, implement and test, and when coupled together will give the functionality of the full system. The method applies both to hardware and software. If the functionality of the modules is clear, the interface between the modules is well defined and cross coupling between many modules are avoided, the modular system is easier to design, implement, maintain and adapt to new requirements. Many different types of modular robots exist. In one extreme, are systems that consist of one basic unit that is able to connect itself to other modules of the same type to form a working robot, as described by Brandt et al. (2007). These are still only interesting on a research level. An example of a successful modular robot system is Lego Mindstorm (Turner 2006), where you based on a few mechanical electronic and software modules are able to build all kinds of working toy robots. This makes Lego Mindstorm a first choice for basic robot education in many places. Other projects also utilise a modular approach such as Elkady et al. (2012) who proposes a three module structure for modular design of sensory modules, actuation platforms and task description. Ertel et al. (2005) presents a modularised flexible robot architecture with a soccer playing robot as example system. Roh et al. (2009) describes a modular architecture that comprises both hardware and software. The architecture is implemented on a personal robot. The most used modular software for robot systems is ROS (Quigley et al. 2009). ROS provides a communication structure between computational units. The basic nodes are tasks that communicate over sockets. The main advantage of ROS is its widespread use, which have lead to a huge amount of solutions available to the community. The downside is performance problems with real-time parts of the system.

Several authors attribute good performance of their real robot systems to the modular design. Corder et al. (2002) describes how they participated and acquired good results in American Association for Artificial Intelligence (AAAI) 2002 Mobile Robot Competition, and that their modular software approach proved useful when using the same autonomous system for different tasks, since large software parts can then be reused. Hong et al. (2017) describes how Korea Advanced Institute of Science and Technology (KAIST) implemented a framework which modularised their software components, in the 2017 MBZIRC competition. This framework allowed them to make successful last minute adjustments to their force-feedback algorithm. The adjustment allowed them to go from a partly successful solution (35 points) to fully successful solution in the Grand Challenge (100 points). Carius et al. (2017) also notes that their modular approach in the same competition allowed them to rapidly adapt to the new environment.

3.1. Mobotware

Our solution to the MBZIRC challenges is implemented in the MobotWare framework as described by Beck et al. (2010). The framework consists of 4 main modules as seen in Figure 3:

- Robot Hardware Demon (RHD): Flexible hardware abstraction layer
- Mobile Robot Controller (MRC): implements and executes basic movement of mobile robots
- Automation Robot Servers (AURS)
- Mission Controller: implements the current mission

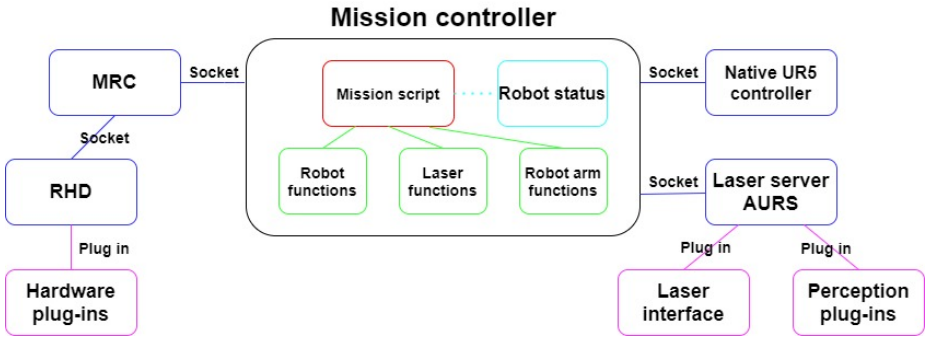


Figure 3. An overview of the mobotware of the UGV.

The modules communicate through sockets and the internal functionality is obtained using function libraries and plug-ins. The most dynamic part is the mission controller, which is implemented using python scripts allowing for fast changes. The use of plugins in the context of mobile robot control is believed to be unique. The plugin concept allows the user to design a basic data and control structure, and then add functionality using plugins. This makes it easier to integrate functionality implemented by different programmers, as the basic structure will remain the same. Another advantage is that the data sharing is more efficient than using sockets. This is important for data intensive sensors, as e.g. laser scanners and vision, where you can have a basic server that takes care of getting the data, and then add functionality writing plugins for the server. Another unique feature is the Mobile Robot Controller (MRC) that takes care of the basic navigation. It contains a real-time interpreter with a scripting language, SMRCL, which makes it possible to adapt the navigation, to a given mission, by putting together basic motion primitives in real-time. We originally tried to use python, but the timing properties of python lead to poor results. The full Mobotware code, as well as, instructions on how to use it, can be found on its wiki page[4].

The MobotWare framework have seen success in multiple projects, such as student projects and other competitions such as the Field Robot competition[5], where the DTU team won a first place in 2018, and have seen good results in other years. During our work with ROS in several master projects e.g. the work of Jensen L. (2019) and Christensen A. (2020), and as external examiner on projects from other universities, we have observed that due to the plugin structure Mobotware solutions tend to have fever tasks (nodes) than ROS solutions. It is a well-known fact, that task switches and intertask communication are less efficient than function calls and access to internal variables. This is supported by the observation that the ROS implementations struggle with timing problems and lack of computer power.

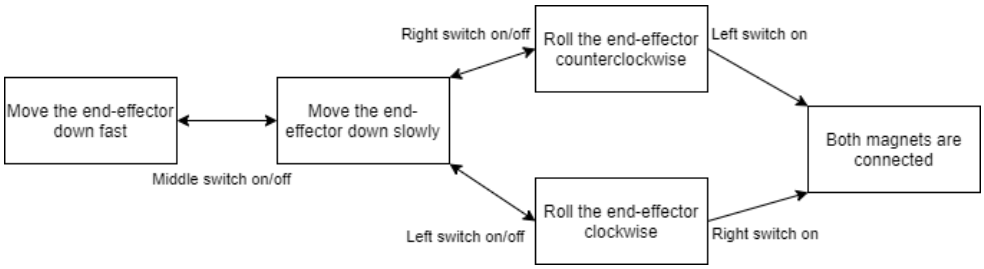
4. Solutions

Due to the difference in platforms few solutions could be used cross platform, and the solutions to the individual platforms were therefore made in parallel. These solutions will be described separately in this section.

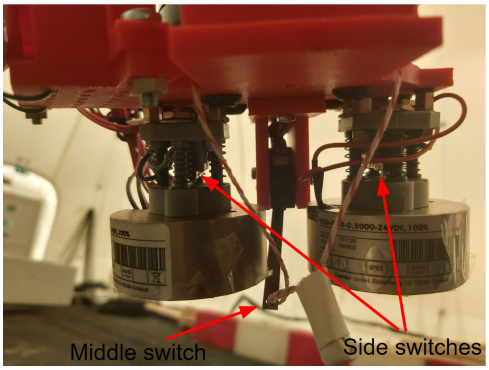
4.1. UGV

The UGV is a SKID steered vehicle with a SICK LMS100-10000 2D-LiDAR-sensor attached to the front. It uses a Universal Robots UR5 as its manipulator, which has a Logitech C922 camera, a Robotiq FT 300 force torque sensor and two Tremba electromagnets attached to it. On top of the

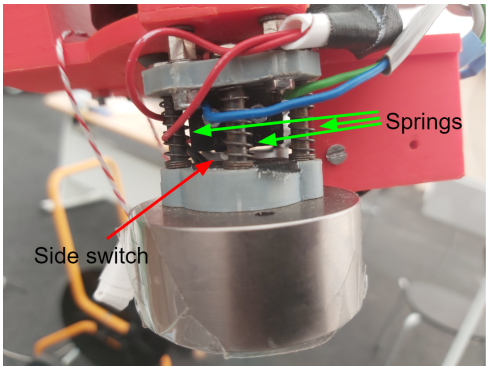
[4] http://rsewiki.elektro.dtu.dk/index.php/Main_Page
[5] https://www.fieldrobot.com/event/index.php/downloads/previous-proceedings/



(a) A flow chart describing the algorithm, which ensures the magnets fully connects to the metal plates.



(b) The front view of the end-effector, showing all three switches.



(c) The side view of the end-effector, which shows how the magnet is separated from the end-effector via springs.

Figure 4. The end-effector and algorithm which the UGV uses to lift blocks. The springs and switches ensures that the magnets fully connect to their target.

UGV a custom built block storage is attached allowing it to carry four red, two green and one blue block while driving without having to use its manipulator. The UGV can be seen in Figure 10.

4.1.1. Mechanics of the arm end-effector

Electromagnets are used as the lifting mechanism of the UGV, and the lifting module is implemented as a robot arm function in Figure 3. A custom built end-effector connects the magnets to the UR5 arm. The end-effector can be seen in Figure 4. The end-effector achieves two objectives: making magnets compliant and providing feedback, in relation to when and how the magnets are connected to the blocks. The magnets are made compliant by separating them from the manipulator by having springs placed in between the magnets and the manipulator. This allows the magnets to adjust to the angle of the metal plate when they are pressed against it. The feedback is achieved by placing switches underneath the magnets. Once a magnet is pushed up into its connected switch it can be assumed that the pressure have caused the magnet to adjust, such that the entire magnet touches the metal plate. Furthermore, the middle switch can be used to detect when a block is dropped, whether it be by accident or on purpose. The compliance and the feedback, obtained by the end-effector, can be combined to ensure the magnets connect successfully. The algorithm achieving this can be seen in Figure 4a.

4.1.2. Vision based block detection

A vision based block detection method can be used by the UGV to obtain a robust block detection, which is independent of the blocks arrangement. This module is implemented as a perception plug-in in Figure 3. The block detection itself was implemented using the state of the art YOLO algorithm, which Redmon et al. (2016) introduced.

(a) YOLO input image with satisfactory object predictions

(b) angleNet input image with satisfactory angle prediction

Figure 5. Examples of predictions from the two networks

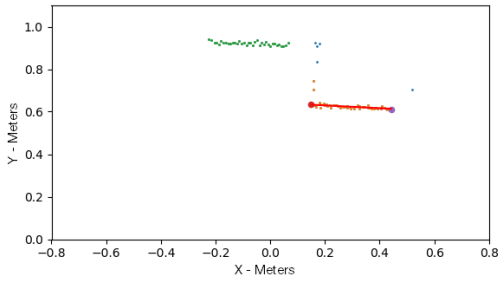
YOLO exists in multiple versions, and the version used in this implementation, *YOLOv2* (YOLO version 2), works by defining bounding boxes (called *anchor boxes*) prior to training and detection. Using the training data and the bounding boxes defined herein, a k-means method is used for determining a collection of predetermined anchor boxes. The last layer of the YOLO algorithm is a convolutional layer with $B(5 + C)$ filters, where C is the number of classes (here 2), B is the number of predefined bounding boxes and 5 is the number of values defining a bounding box; x,y-coordinates of the center, w (width) and h (height) as well as p_{obj} which is the likelihood that a grid cell contains an object. The resulting activation map is $N \times N \times B$ and represents a grid of predictions for each bounding box with a center in each grid cell. Thus, for each grid cell for each anchor box, the algorithm calculates the predicted confidence for each class. The architecture of the YOLO network is described by Redmon and Farhadi (2017), and its implementation is available as shown by Huynh (2019). To estimate the orientation of the brick w.r.t. the ground plane, an angle estimation regression network (*angleNet*) is trained on the same data as the YOLO network. To estimate the angle of a detected block, the bounding box is cropped from the detected image and pasted on an empty image of the scene. The resulting image thus contains only one block. The resulting networks is capable of detecting one or more blocks in an image. Figure 5 shows examples of satisfactory predictions. Figure 5.b shows examples of the input to the angleNet, as well as, the predicted angle and x,y pixel coordinates.

4.1.3. Laser based block detection

The UGV can utilise a laser scanner based block detection, implemented as a laser function in Figure 3, in addition to the vision based block detection. The laser based block detection takes advantage of the fact that the blocks are stacked neatly in predefined positions relative to each other. Each stack consists of smaller stacks, which we will denote pillars. This method detects the pillars by clustering laser points together. The max distance between cluster points is defined using the predefined knowledge of the distance between the pillars. The position of middle of the block can be calculated once the edge of the block are found, since the size of all the blocks are known. Figure 6b shows a pseudo code implementation of the algorithm. Figure 6a shows the analysed laser scan of a red stack. One pillar have already been completely removed from the stack. The orange and green dots are two separate clusters, which also corresponds to two different pillars. The estimation of the corners of the pillar are shown by the two larger dots and the estimation of the angle of the pillar is shown as a red line from corner to corner.

4.1.4. Stack navigation

The UGV is able to navigate to the back of stack using only predefined instruction. Largely the same laser detection method, as described in section 4.1.3, is used to identify the stacks. Small modifications are made to make it identify the blue stack instead. The modifications can be seen in

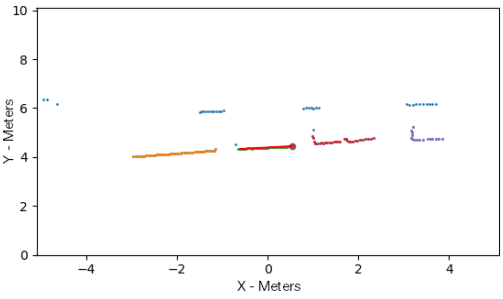


(a) An analysed laser scan of the red stack after a group red of blocks (pillar) have been removed. The red line is the estimated angle of the group of red blocks, which the UGV is about to pick up.

```
def detect_block_laser(block_color):
    threshold = DISTANCE_BETWEEN_PILLARS
    # Get clusters sorted from close to far
    clusters = get_laser_clusters(threshold)

    # Get the closest cluster
    closest_cluster = clusters[0]

    # Get the x and y coordinate and
    # rotation of the desired object
    x, y, rotation = calculate_middle_pos(closest_cluster,
                                          block_color)

    return x, y, rotation
```

(b) Pseudo code describing the laser based block detection.

Figure 6. The laser based block detection.



(a) An analysed laser scan of the stacks, with the blue stack correctly identified. The estimated angle of the blue stack, and therefore all stacks, can be seen as the red line.

```
def detect_stack_laser():
    threshold = DISTANCE_BETWEEN_STACKS
    # Get clusters sorted from close to far
    clusters = get_laser_clusters(threshold)

    # Get the 4 closest clusters
    closest_clusters = clusters[:4]

    # Sort the clusters from right to left
    sorted_clusters = sort_clusters(closest_clusters,
                                    direction=RIGHT_TO_LEFT)

    # Get the cluster representing the blue stack
    blue_stack = sorted_clusters[1]

    # Get the x and y coordinate
    # and rotation of the blue block
    x, y, rotation = calculate_middle_pos(blue_stack,
                                          BLUE_BLOCK)

    return x, y, rotation
```

(b) Pseudo code describing the laser based stack detection.

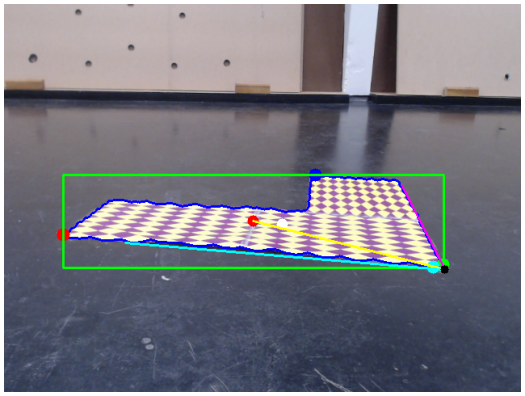
Figure 7. The laser based stack detection used for stack navigation.

Figure 7a. The blue stack is used to estimate the angle of the stacks compared to the UGV, since it is closest to the middle of all the stacks and it is a single block stack. Figure 7b shows the laser scan of the stacks, with the blue stack correctly identified. Since the relative position among the stacks are predefined, the UGV is able to navigate to all the stacks in open loop, once the position of the blue stack is found. When the UGV is in front of a stack, it drives the remaining distance in a closed loop, to reduce uncertainties. The stack detection is implemented as a laser function, while the driving part is a robot function in Figure 3.

4.1.5. Wall navigation

The UGV locates the wall with two perception plug-ins. The first plug-in locates the "L" in a rough manner, and the second plug-in estimates the orientation of the "L". The UGV can use the position of the "L" to navigate to the left side of the "L". The sides of of the "L" will be called "legs". From the left "leg" the UGV can start placing the blocks on the wall.

The first plug-in works by utilising the fact that, the checkers pattern is significantly lighter than the dark asphalt. This means that the wall marking can be detected by applying a mask to the image which sorts the pixels into dark and light pixels. By choosing the largest contour, other light disturbances in the image are able to be filtered out. The UGV can then use the position of the largest contour as feedback in a loop, which allows the UGV to navigate close to the "L". Figure 8b



(a) An analysed image of a mock-up of the "L", where the left and right "legs" are detected.

```
def detect_left_leg(largest_contour):
    # Get the "legs" of the "L"
    hough_lines = get_hough(largest_contour)

    # Define leg 1 as the longest line
    L1 = longest_line(hough_lines)

    # Define leg 2 as the longest line with
    # an angle which is not close to leg 1
    L2 = longest_second_line(hough_lines, L1)

    # Define the corner as the position where the legs cross
    corner = cross(L1, L2)

    # Define the splitting line as a line from the center of mass
    # to the corner of the "L"
    splitting_line = get_splitting_line(center_of_mass, corner)

    # Test which leg is the right leg by checking
    # if it is earlier on the unit circle
    if leg_earlier_than(L1, splitting_line):
        left_leg = L1
        right_leg = L2
    else:
        left_leg = L2
        right_leg = L1

    return left_leg
```

(b) Pseudo code describing the how the "L" is analysed.

Figure 8. How the left "leg" of the "L" is found, which is used as a starting position for the UGV when building the wall.

shows the pseudo code of the second plug-in, which is able to determine the position of the left "leg". Figure 8a shows an image of an "L" with the algorithm applied to it. The magenta and cyan lines are the two detected "legs", and the yellow line is the line which splits the "L" in two. The green box is a bounding box encasing the largest contour in the image.

4.1.6. Avoiding block collision

The module which places blocks on the wall, is a Robot arm function as seen on Figure 3. To avoid colliding the blocks, the UGV deliberately overshoots the placements of the blocks. This results in gaps between the blocks creating holes in the wall. These holes makes it more difficult to estimate the wall, compared to if it was one whole object, and they also pose a risk of the smaller blocks falling over if placed partly on top of a hole. Therefore, the UGV does not immediately place the block down on the wall. Instead the UGV hovers the block slightly above the surface below and moves the block towards the former placed block. The force torque sensor is used to detect when the two blocks touch each other, and by doing this the UGV can ensure that the wall contains no or only small holes.

4.1.7. Wall building

As mentioned in section 2, the UGV must build two sides of the wall, where one is completely orange, and the other is a random pattern of colour.

The first orange block is placed by using visual feedback. This is done by detecting the edges of the left "leg", since the size of the leg is known. The edges are detected by applying a mask to the image similar to the mask used in the "L" detection. The extreme points of the obtained contour is then assumed to be the position of the edge of the "L". This can be done since the UGV is aligned with the left "leg". This is realised as a perception plug-in in Figure 3. The subsequent orange blocks are placed by detecting where the first orange block was placed and placing the next orange blocks on top. The laser scanner and the laser module described in Section 4.1.3, are used to detect the first orange block.

Once the orange block has been placed, the UGV starts building the coloured wall. The UGV uses the starting edge of the coloured wall as a reference point. From that reference point the UGV relies on its odometry to determine when to stop to place the blocks at the right spots. When building the first coloured layer, the UGV continues to use the visual detection, described above, to estimate how far in to place the blocks, as well as the force feedback to place the blocks next to each other.

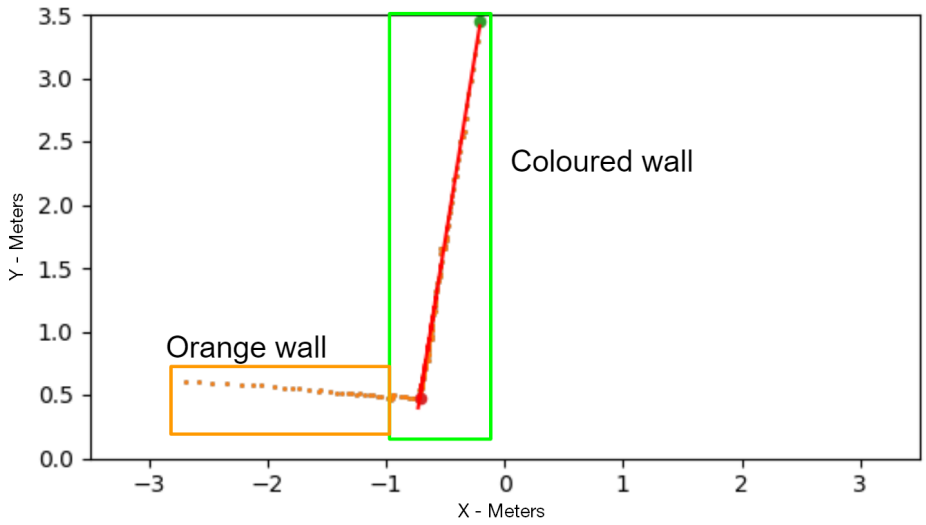
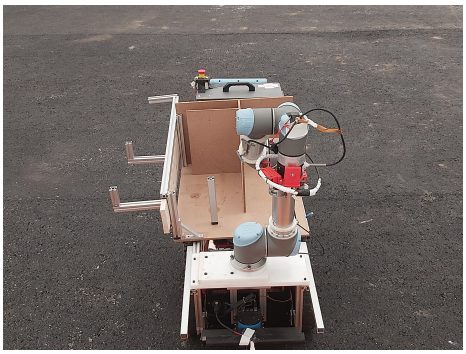
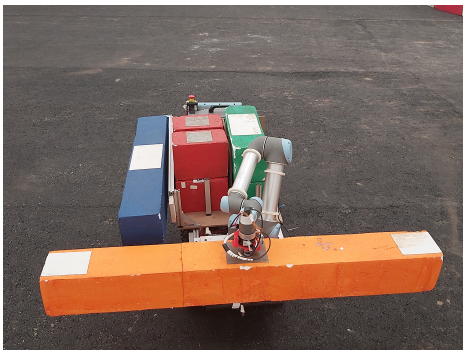


Figure 9. An analysed laser scan of entire wall with the wall segments marked.



(a) The unloaded block storage of the UGV.



(b) The loaded block storage of the UGV.

Figure 10. The block storage of the UGV loaded and unloaded.

When placing the subsequent layers, the UGV uses laser scans of the wall to determine how far away it is from the wall and how far in to place the blocks.

Figure 9 shows how the UGV detects the coloured wall. The entire wall is found as a single cluster, and the right side of the "L" is used as the coloured wall. The angle of the wall can then be calculated, which the UGV uses as feedback to drive next to the wall. The actual orange and coloured part of the wall is marked with boxes.

4.1.8. Block storage

Figure 10 shows how the UGV stores the blocks. The UGV is able to store four red, two green and one blue block at time. It also has two metal bars to rest the orange block on while driving, this decreases the risk of dropping the orange block. As it can be seen on Figure 10b, the blocks are packed tightly, and they are packed more tight than the error on the block detection allows. The UGV is able to do this due to the force sensor on top of its end-effector, which can detect when the blocks are pressed against the storage walls.

The block storage is designed in such a way that any colour can be lifted or placed at any time, making it independent of any pick up or build order. The block storage module is a Robot arm function in Figure 3

Figure 11. The UAV in contact point with a red block.

4.2. UAV

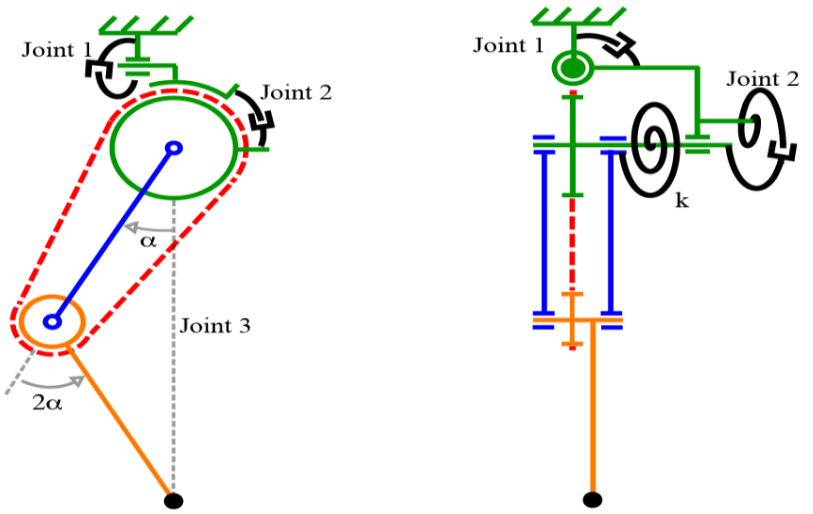
The UAV is a DJI M100 quadrotor in X configuration. It is equipped with a TeraRanger One rangefinder for measuring the current flying height. It uses a Raspberry Pi V2.1 camera module mounted on a 2 axis gimbal for visual block detection. For picking up and flying with blocks it has a custom designed arm with electromagnets mounted on the end. Lastly it uses a Raspberry Pi 3 as its onboard computer(OBC), which is used for position control, mission handling and image analysis. An image of the UAV can be seen in Figure 11.

4.2.1. Mechanics of the arm end-effector

The manipulator of the UAV is a 2-links manipulator allowing compliant passive motion on the linear direction between the end-effector and the base joints (Joint3 in Figure 12) and damped passive rotation at the base revolute joints (Joint 1 and Joint 2 in Figure 12). At the base of the manipulator 2 free-joints are mounted, providing a purely damped rotational connection between the base and the manipulator's rotation. This allows the manipulator's rotation to be decoupled from the aerial platform rotation, which consequently allows for increased stability of the overall system at contact. The linear compliant joint has been achieved by designing a two-link manipulator, and by coupling the rotation of the two links via a belt, so that the distal link counter-rotates twice as much as the proximal link. A torsional spring connects the proximal link to the 2-DoF rotational base joint. The base joint consists of a series of two passive revolute joints where the joint rotation is damped. This allows the manipulator to be fully passive towards physical interaction, which is a necessary condition to minimise the disturbances introduced on the aerial platform when the end-effector interacts with the environment. The damping mechanism further allows for a reduction of the oscillations of the manipulator and the grasped block during free flight, this allowing for increased precision of the placement of the block and an improved flight control. The damping mechanism has been obtained by using unconnected electric motors.

At the endpoint, a small end-effector is attached on a freely rotating axis. The tool consists of 2 electromagnets mounted side-by-side with a small tactile switch in between. The magnets are used to hold on to the block and the switch provides a feedback to determine when a block has been touched and if it is still mounted on the arm.

With a manipulator like this we achieve an arm, that is mechanically compliant enough, that it reduces the requirements on the drones precision. It also ensures that the UAV remains stable even



(a) side view of the manipulator's sketch design

(b) front view of the manipulator's sketch design

Figure 12. Sketch design of the manipulator. Three passive joints in a R-R-T configuration, where the two rotationsl joints (R) are passive through a damping system, and the translational degree of freedom (T) is passive and compliant. The specific structure uses the concept proposed by Fumagalli et al. (2016)

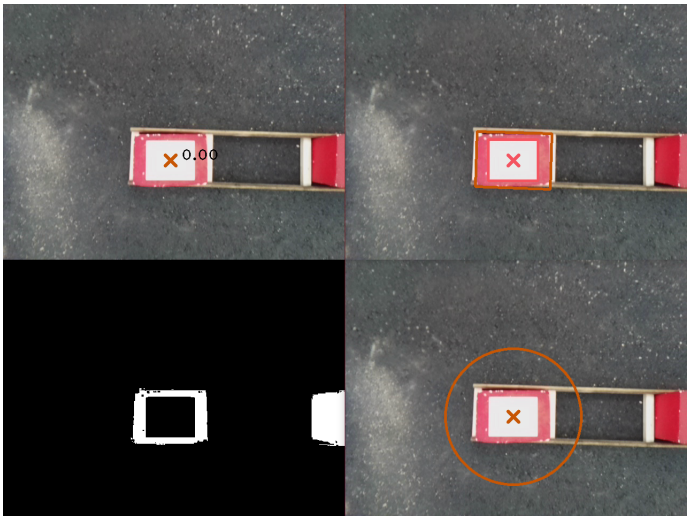


Figure 13. (top-left) Center of a detected block with estimated angle (top-right) The estimated inner and outer rectangles with their center (bottom-left) The thresholded image based on red (bottom-right) The region of interest that will be searched in the next image

when attached to a block while the block is still on the ground. A very uncompliant arm would drastically change the dynamics of the drone during this contact phase and when flying with the block.

4.2.2. Vision based block detection

By having a camera mounted facing downwards it is possible to make a broad search area, and when a block is detected, act on this information e.g. fly down towards it and pick it up. A detected block, with angle and center detected, can be seen on Figure 13 top-left.

Once an image is captured it is converted to an HSV (hue, saturation, value) image and blurred. A min/max normalisation is applied to reduce the effects of external factors such as lighting. The image is then thresholded based on the colour of the block, that is being searched for. Two morphological transforms are next performed on the thresholded image. We perform *opening* and *closing* of the threshold image to remove noise and smooth out the thresholded features, so they become better closed off contours. An example of the thresholded image can be seen on Figure 13 bottom-left.

Using the thresholded image, the contours are found in the mask image. Small contours are discarded. Next we use the hierarchy of contours to find the block. Because the block have a white metal plate, the thresholded image of the block will have an outer and inner contour. By utilising that the block will have both an inner and outer contour we filter valid blocks from other contours. With the valid contours left, the minimum enclosing rectangle is estimated and used to find the center, as well as, the angle of the rectangle and thus the angle of the block. The estimated rectangles can be seen on Figure 13 top-right.

After a block is identified, a region of interest is selected around that block, and in the subsequent images, the region of interest is used when detecting the block. This will ensure that we keep track of the same block when multiple blocks could be visible. The region of interest is exemplified on Figure 13 bottom-right.

If the block is only partially visible, which will be the case when trying to pick up the block as it is to large to be in the image, it is necessary to still be able to track the position and angle. Because of how the drone approaches the block, only the top half of the block will be visible. By identifying the top left corner of the block and the white plate in the image, the vector going from a corner to the other corner can be used to calculate where the center of the block should be, because of the fixed geometry of the blocks.

4.2.3. Navigation and localisation

To keep track of the drones position, we use a simple physical model of motion based on accelerometer- and gyroscope-data combined with compass-, GPS- and visual-data. Using a discrete state model with, an observer, the drone is able to keep a steady estimate of its current position, that can be used in combination with the DJI flight controller.

Using the accelerometer and gyroscope, a short term position estimation is possible by filtering and integrating these sensor data. This is run at a constant rate with a state space model. To correct for drifts caused by measurement noise and bias, a linear observer is included to correct the position estimates based on the different positioning scheme used. By using this approach it is easy to change between different positioning modules, as the position controller is independent on what localisation module is used, as long as the coordinate frame is right handed and has x forward and z up.

Two localisation modules was needed for Challenge 2: A GPS based one for general movement around the field and a specific one for block pickup.

4.2.4. GPS based navigation

When navigating around the challenge arena the UAV uses its GPS to maintain a steady position. The GPS is not precise enough for finer tasks, but for general navigation around the course it is sufficient to use as odometry.

Once the UAV starts a mission it saves the GPS longitude and latitude of the takeoff position, as well as, the compass heading. This is saved as the origin for the odometry for the rest of the flight. The current heading is set as 0° and the odometry system is defined as a right hand coordinate system with the x-axis pointing in the direction of 0°.

4.2.5. Visual (block) based navigation

When the UAV needs to pick up a block, the block becomes treated as a fixed frame of reference that the UAV can navigate relative to. The coordinate system on the block is defined by having

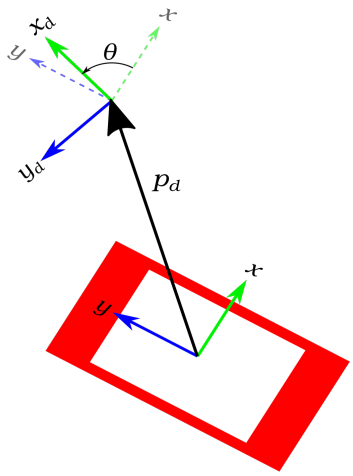


Figure 14. Figure depicting the block's coordinate system in relation to the UAV's coordinate system.

the x-axis pointing out of the short side of the block and having the y-axis pointing out of the long side of the block, as seen on Figure 14. From the camera based block estimation, which provides the center of the block, as well as, the rotation of the block, it is possible to convert this to a xy-position and a θ-rotation (yaw). Based on Figure 14 it can then be extracted what the position of the drone, p_d, is in relation to the block as well as its heading, θ.

It is convenient to calculate the position like this, as it is known that the blocks are placed on the ground plane, meaning that the image plane and ground plane are parallel. As accurate height measurements are available from the laser rangefinder, it is possible to get a good estimate of the center of the block by using the pinhole camera model. The position of the block in the drone frame is calculated as:

$$\frac{x_i}{f} = \frac{x_b}{z_d} \tag{1}$$

$$x_b = \frac{x_i z_d}{f} \tag{2}$$

where f is the focal length of the camera, x_i is the pixel coordinate of the center of the block, z_d is the height of the drone above the top of the block and x_b is the position of the center of the block in the real world. The same calculations can be performed on the y axis to get the 2D position. Using a simple yaw-rotation the position of the drone in the block frame can be found:

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} x_b \\ y_b \end{bmatrix} \tag{3}$$

5. Competition performance

5.1. UGV

As mentioned in Section 2.3, the UGV was supposed to build multiple layers. This did not happen during the competition, due to an unexpected problem, which appeared during the rehearsal days. The magnetic properties of the metal plates of the real blocks were significantly less than the ones of our own mock-up blocks. This resulted in an overestimating of the UGV's ability to lift the blocks. It was decided that the margin of error for lifting the blue and orange blocks were too high, and a new strategy therefore had to be formed. The goal of the UGV became instead to ensure that it would still gain autonomous points. Under the real conditions the green blocks were the ones which had

the highest success rate of being packed correctly. The strategy therefore changed to only placing green blocks, similar to the UAV's strategy. The problem was first fully realised after the second trial attempt, meaning that time until the competition was short, which was the reason for the large downgrade in ambition. Due to the modular configuration of the UGV, it was possible to change the system enough, that the UGV could directly place a green block on the first competition day, and through additional improvements two green blocks was placed on the second competition day. The final wall, which was enough to secure a third place in Challenge 2, was therefore two green blocks placed next to each other.

5.2. UAV

During the rehearsal days, measurements of the course were made so that approximate locations of the blocks and wall could be noted. From these measurements it was possible to write a mission script, that could handle flying to the approximate location of the blocks and then search the area, to locate a suitable block for pickup. When a block was detected, it was a simple task to switch the control to the block pickup module that would then attempt to perform the pickup.

After a successful pickup the UAV was able to fly towards and past the wall in an attempt to drop the block on the wall, and then return to the blocks and repeat this process. During the competition it successfully picked up three blocks and flew them towards the wall, but because of wall detection malfunctions all three were unfortunately dropped before the wall was reached. Due to the former cooperation strategy, development on the block delivery system was started late and was therefore the most underdeveloped part of the UAV, and the UAV, therefore, received false height measurements, which was not handled. This problem was fixed after the competition.

Due to the above mentioned problem, the UAV was unable to score any points in Challenge 2, but the competition performance can still be considered a partial success, as the UAV was able to autonomously pick up multiple blocks.

5.3. Evaluation

The modules allowed for the mission scripts to be written at the deployment site, once the site had been surveyed. The UVs was not able to execute the main strategy, and did therefore not perform as hoped. There were valid reasons for their shortcomings, as mentioned above, and the final result was therefore not disappointing. The time constrained deployment window was not the limiting factor in the success of the UVs, proving the usefulness of a modular approach. It was in fact the modular approach which allowed the UGV to be modified to such a degree, that it was still able to obtain points, even though it could not fulfil its original strategy.

The final score of our systems was 0.89, which puts it in close proximity to the 2. place. Only 5 out of the 22 teams, who participated in Challenge 2, where able to score any autonomous points. It is worth noting that the winning team scored high points using mostly UAV, and the second place team scored points exclusively using their UAV. Our UGV solution scored the highest amount of points out of any of the UGV's. The complete score table can be found on the MBZIRC website[6].

5.4. Further testing at home laboratory

Additional tests were conducted at home, both prior and after MBZIRC. Using a setup, similar to the setup described in Section 2, the UGV is able to consistently build one coloured layer and half of an orange layer, which would result in 15 points, almost double that of the winner of the challenge. Additionally, the UGV can also build the second layer gaining another 15 points. However, the UGV sometimes fail when building the second layer. The failures happens because the UGV

[6] https://web.archive.org/web/20210514102710/https://www.mbzirc.com/winning-teams/2020/challenge2

sometimes collides the blocks when building the second layer. These collisions happens due to a lack of feedback of the blocks above the ground, as the laser scanner can only detect blocks at ground level. We only have enough mock-up blocks for two layers, and only two layer tests have therefore been conducted. After the competition, the delivery module of the UAV was finalised, and the UAV is able to consistently deliver red blocks to a wall. This indicates that the methods used for the UAV was successful, but underdeveloped, and the main shortcoming of the UAV, was therefore lack of development time. A video of the UGV successfully building two layers in our home laboratory can be watched in this video[7]. Additionally, the drone can be viewed placing two red blocks on our mock-up wall in these videos[8][9]. The drone is unfortunately no longer in a state where additional tests can be conducted, but our anecdotal experiences suggest a successful pickup rate of roughly 80% and a delivery rate of around 30%.

The block-lifting and storage module of the UGV were additionally tested by having the UGV place blocks from its storage module in the same manner as the stacks would look. The blocks was then lifted up again using the block detection module and placed in the storage. The whole process was then repeated. It is worth noting the UGV placed the blocks in stacks which was significantly worse than they were at the competition, meaning this scenario is harder than the real one. Each colour of the blocks was tested individually, and the UGV was able to successfully lift and fill up its storage 200 times in a row for each colour. This shows that the block handling modules of the UGV are robust, and it also mirrors the observation that the weakest part of the UGV is building the wall.

The wall building was tested by having the UGV build a wall from a fully loaded state. Both the first layer and the second layer building were tested, as these represents two different states for the UGV. In total 20 runs for each layer were executed. The UGV was able to successfully build the first layer 20 out of 20 times and 18 out of 20 times on the second layer. This matches our expectation of the UGV, as mentioned above.

Before the competition, the cooperation strategy mentioned in Section 2.3 was tested. The cooperation strategy took advantage of the speed of the drones and the precision of the stable platform of the UGV, and early tests were successful, as seen in the Cooperation Test video[10].

6. Discussion

Several lessons were learned from participating in MBZIRC 2020. The most valuable or noteworthy ones, will be described in this section.

Having multiple independent and self-correcting modules reduces the uncertainties of a system, which would otherwise be additive and could become large enough to result failure. This proved true at the competition. The UGV had managed to pack two green blocks in its block storage, but the top one had been placed incorrectly, and was leaning on the side of the container. Despite the block being tilted around the horizontal axis, the UGV was still able to lift it from its block storage due to the automatic adjustment from the end-effector, which effectively saved the run which resulted in a third place. This lesson is also enforced by the block lifting experiment, described in Section 5.4, as the robustness of the block handling modules can be contributed to the multiple independent and self-correcting modules.

Having redundant modules, while expensive, can improve the likelihood of success. The main reason the UGV was unable to score the desired amount of points, was due to its inability to lift the blocks. If another lifting module had been designed, then that module could have been swapped in and the UGV might have been able to score enough to win Challenge 2. Such redundant modules

[7] https://www.youtube.com/watch?v=VblUV8BAKHY
[8] https://youtu.be/vUgBRVFPNRI
[9] https://youtu.be/qcZo3ICK_SU
[10] https://youtu.be/n1RWnoXez9g?t=299

was made for the block detection, in the form of both a laser and image based block detection, and it made it possible to choose the one which fitted the task best.

The reusability of the modules saves time and cost during the development. We saw good results from reusing the same modules for different tasks, as the same modular building blocks, which was used to complete Challenge 2, was used to create the mission script for Challenge 3, where the UGV obtained a second place.

Štibinger, S. et al. (2021), who won Challenge 2 and the Grand Challenge of MBZIRC 2020, and Lenz, C. & Schwarz, M. et al. (2020), who came second in Challenge 2 and won the Grand Challenge of MBZIRC 2017, notes that they had trouble successfully deploying their full UGV solution, due to the unexpected environment changes, in the form of a slope, and the limited time allowed for testing and deployment. Štibinger, S. et al. (2021) was able to place a single red block using their UGV, and Lenz, C. & Schwarz, M. et al. was unable to place any, with their UGV. This shows the difficulty of deploying an autonomous system when the environment can change unexpectedly, and while a time restraint is present. These problems also affected our solution, but our UGV was able to score the most points out of any of the UGV solutions, and we believe having simple modules, which are easy to adjust, is what allowed us to achieve the rapid deployment of our autonomous systems. We scored high points already on the first competition day, especially in Challenge 3, where our full UGV solution was able to be fully deployed.

An alternative to utilising modules and reconfiguring the system at the deployment site, could be create multiple programs for the same system, with varying degrees of complexity, an e.g. could be having a full program and an emergency program. The full program would be the full solution, and the emergency program would try to only fulfil critical parts of the task. Štibinger notes that they were unable to deploy their full solution, and therefore went with their emergency protocol. An upside to a fully ready emergency program, compared to having to assemble it using modules, is that it would be ready to use immediately, whereas the modular solution takes time to assemble, even if it is a small amount. This is assuming that the emergency program works out of the box, and a modular solution can still be effective, depending on the time allowed for deployment, as the modular solution's effectiveness improves with the time available, and the choice of method should therefore depend on the situation.

Our solution was made to solve the challenges of the competition, and as such is not a fully fledged solution, which would be applicable in real world scenarios. However, the underlying modular approach showed promising results in solving the parts of the challenges, which also appears in the real world. The challenges contained different unexpected problems, which the modular approach was able to alleviate. These type of unknown and unexpected problems are an even larger problem in real world scenarios, and as such the need for a solution to them is even greater, showing the potential of modular solutions.

7. Conclusion

In this paper a modular solution to Challenge 2 of MBZIRC is described. The implementation of the solution and the reasoning behind the design choices are included. The solution includes both UGV and UAV solutions, and an evaluation of their effectiveness.

Our UVs participated in the MBZIRC 2020 in Abu Dhabi, where the effectiveness of the modular solutions could be tested and lessons could be learned. Testing time on the actual deployment site was limited, and unexpected challenges, such as a slope, made the challenge more difficult than predicted. In particular, the magnetic properties of the blocks was weaker than we had foreseen. This meant that our full solution could not be deployed, but due to the modular design of the UGV, we were able to construct a backup solution. This was done despite the limited time and being outside our lab environment. This backup solution was a large downgrade in ambition, but it was still able to score the most points of any of the UGVs, showcasing the capabilities of a modular design. Overall our systems were able to obtain a fourth place in the Grand Challenge, a third place in Challenge 2 and a second place in Challenge 3.

ORCID

Tobias Stenbock Andersen https://orcid.org/0000-0002-3976-8987
Matteo Fumagalli https://orcid.org/0000-0003-1485-4616
Ole Ravn https://orcid.org/0000-0003-2265-0031
Nils Axel Andersen https://orcid.org/0000-0002-9985-6444

References

Beck, A. B., Andersen, N. A., Andersen, J. C., & Ravn, O. (2010). Mobotware – A Plug-in Based Framework For Mobile Robots. In *IAV 2010 International Federation of Automatic Control*. (pp. 127-132). doi: 10.3182/20100906-3-IT-2019.00024.

Boehm, B. W. (1988). A spiral model of software development and enhancement. In *Computer, vol. 21, no. 5*. (pp. 61-72). doi: 10.1109/2.59.

Brandt, D., Christensen, D. J., & Lund, H. H. (2007). ATRON robots: Versatility from self-reconfigurable modules. In *International Conference on Mechatronics and Automation, 2007, Harbin, China*. (pp. 26-32). doi: 10.1109/ICMA.2007.4303511.

Carius, J., Wermelinger, M., Rajasekaran, B., Holtmann, K., & Hutter, M. (2018). Deployment of an autonomous mobile manipulator at MBZIRC. In *Journal of Field Robotics. 35*. (pp. 1342–1357). doi: 10.1002/rob.21825.

Christensen, A. (2020). Comparative study of middleware for mobile robot control (Master Thesis). *Department of Electrical Engineering at the Technical University of Denmark*

Corder, J., Hsu, O., Stout, A., & Maxwell, B. (2002). A Modular Software Architecture for Heterogeneous Robot Tasks. In *AAAI Mobile Robot Competition & Exhibition Workshop*. (pp. 18-23).

Elkady, A., Joy, J., Sobh, T., & Valavanis, Kimon. (2012). A Structured Approach for Modular Design in Robotics and Automation Environments. In *Journal of Intelligent and Robotic Systems. 72*. (pp. 5-19). doi: 10.1007/s10846-012-9798-y.

Ertel, W., Fessler, J., & Hochgeschwender, N. (2005). A universal modular autonomous robot architecture. In *Proceedings of the Second International Conference on Informatics in Control, 2005, Barcelona, Spain*. (pp. 391-394).

Fumagalli, M., Barrett, E., Stramigioli, S., & Carloni, R. (2016). Analysis of an underactuated robotic finger with variable pinch and closure grasp stiffness. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. (pp. 365-370). doi: 10.1109/AIM.2016.7576794.

Hong, J., Jung, S., Jung, C., Jung, J., & Shim, D. (2018). A general-purpose task execution framework for manipulation mission of the 2017 Mohamed Bin Zayed International Robotics Challenge: HONG et al.. In *Journal of Field Robotics. 36*. (pp. 149–169). doi: 10.1002/rob.21820.

Jensen, L. (2019). Navigation for Mobile Robots Using ROS (Master Thesis). *Department of Electrical Engineering at the Technical University of Denmark*

Lenz, C. & Schwarz, M. et al. (2020). Autonomous Wall Building with a UGV-UAV Team at MBZIRC 2020. In *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. (pp. 189-196). doi: 10.1109/SSRR50563.2020.9292580.

Huynh, N. A. (2019 December 31) *YOLOv2 in Keras and Applications*. Retrieved from https://github.com/experiencor/keras-yolo2.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software. 3*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, Las Vegas, NV, USA*. (pp. 779-788). doi: 10.1109/CVPR.2016.91.

Redmon, Joseph & Farhadi, Ali. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, Honolulu, HI, USA*. (pp. 6517-6525). doi: 10.1109/CVPR.2017.690.

Roh, S., Yang, K., Park, J., Moon, H., Kim, H., Lee, H., & Choi, H. (2009). A Modularized Personal Robot DRP I: Design and Implementation. In *IEEE Transactions on Robotics, vol. 25, no. 2*. (pp. 414 - 425). doi: 10.1109/TRO.2009.2014499.

Royce, W. W. (1970). Managing the development of large software systems. In *Technical Papers of Western Electronic Show and Convention (WesCon), 1970, Los Angeles, USA*. (pp. 328-388)

Štibinger, S. et al. (2021). Mobile Manipulator for Autonomous Localization, Grasping and Precise Placement of Construction Material in a Semi-Structured Environment. In *IEEE Robotics and Automation Letters, vol. 6, no. 2*. (pp. 2595-2602). doi: 10.1109/LRA.2021.3061377.
Turner, D. (2006). Lego Mindstorms NXT. In *Technology Review. 109(3)*. (pp. 22–23).

How to cite this article: Andersen, T. S., Nielsen, A. N. H., Fumagalli, M., Axelsen, J. L., Christiansen, M., Ravn, O., & Andersen, N. A. (2022). Modular design and implementation for rapid deployment of autonomous systems. *Field Robotics, 2*, 1951–1970.