BioMAV: the package-inspecting robot

Robin Wellner and Roland Meertens

May 13, 2013

1 Introduction

BioMAV (which stands for "Biologically inspired Micro Air Vehicles") is a recurring project of the department of Artificial Intelligence at the Radboud University in Nijmegen. This report contains an overview of this years project, the approaches taken, materials used and the resulting program.

1.1 Drone

The robot of this project is the Parrot AR.Drone 1.0. It is able to communicate with a laptop using USB or Wi-Fi. The drone has two cameras, one ground-facing and one forward-facing. However, the driver software used was only able to access the image data from the forward-facing camera.

1.2 ROS

ROS (for Robot Operating System) is an open-source framework for robot controller software and is especially intended for the re-use of libraries (called "nodes") between different projects and for different types of robots. This allowed us to make use of existing solutions for computer vision, drone control and inter-node communication.

1.3 Vision

For the processing of camera images the CMVision library was used. This library contains several computer vision techniques which apply low level image processing techniques to segregate blobs from images. More about the vision component can be found in section 3.1

2 Task

The goals of this project were:

1. To establish a platform, based on the old BioMAV project, on which future BioMAV projects can build.
2. To let the drone perform a task autonomously.

Figure 1: A picture of a drone.

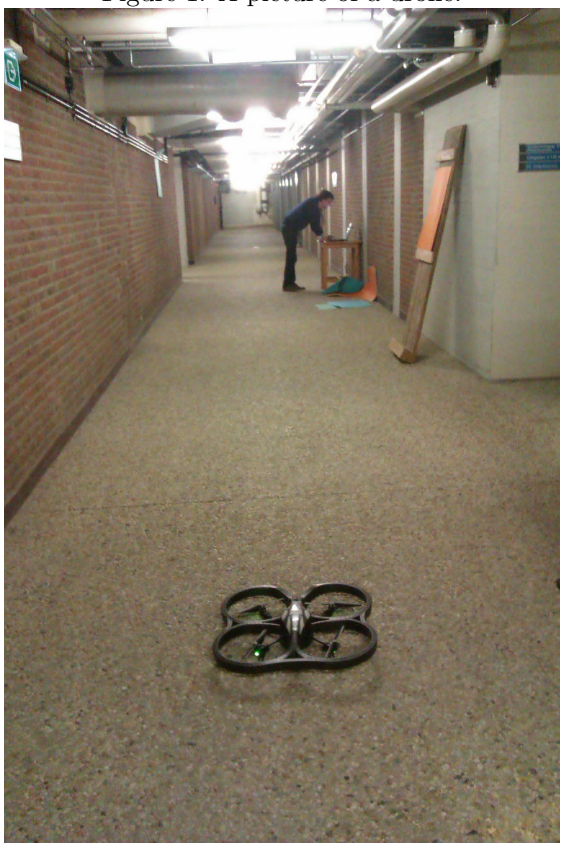


Figure 2: A picture of a drone.

3. To demonstrate the drone performing that task.

The task that the drone had to perform was a search and deploy task. In this task the robot searches for a goal where it has to perform a simple task. This led to task where the drone inspects a package by flying towards this package.

Initially a fly-to-object behaviour was created. As this does not yet give way to very advanced behaviour an implementation for a breadcrumbs finding navigation task was chosen as task.

In this breadcrumbs navigation task the drone is able to locate a goal object by following other objects. Every time a temporary goal has been reached it will start searching for the next goal.

3 Method

3.1 Blob detection

3.1.1 Using CMVision

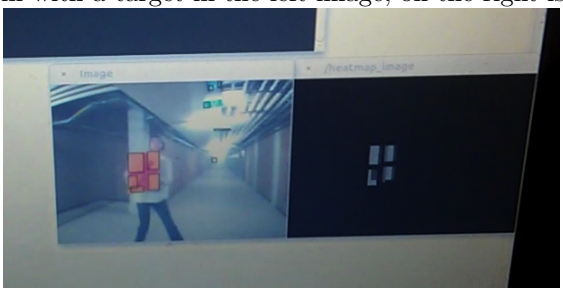
We found a ROS package called CMVision that could find rectangular blobs of roughly the same color in images, and could do that in real time. We used it for every task to both find targets and were even able to recognise corridors by looking at the fluorescent lamps.

CMVision comes with a tool, called colorgui, which allows one to calibrate the range of values used to identify blobs. The dramatic influence of environment lights on the colors meant that each time the environment changed the colorgui tool had to be used again to recalibrate the colors.

In our initial setup (with ROS Electric), CMVision did not specify correctly which color each blob was. That meant a few functions had to be implemented that manually checked the colors of the camera pixel values.

3.2 Heat map

Figure 3: Robin with a target in the left image, on the right is the activation



3.2.1 Idea

At some point, it became clear that the blob detection was not consistent. Small blobs were often found where none should be, and the blob corresponding to the target fell away for a frame every so often. This confused our software, making navigation less than optimal.

As a solution a heat map was implemented which would only activate after blobs appeared consistently on a certain position. At the same time, a single missed frame wouldn't mean that the drone had lost its target.

3.2.2 How it works

The heat map is a matrix, each cell corresponding to a square of pixels from the drone camera. How many pixels depends on the "downsize factor", which reduces the size of the matrix to deal with CPU-heavy calculations. Every blob detected increases the values in the heat map inside that blob. Each time step, the matrix is multiplied by a constant $c_{cooldown}$, where $0 < c_{cooldown} < 1$, reducing the overall activation.

Only if the maximal value in the matrix is more than a specified constant, the drone considers doing anything based on the heat map that time step. During control, when there is no activation it will turn around slowly, looking for the target.

If the sum of all values in the matrix is more than another specified constant, that means a large portion of the camera's field of vision is taken up by the target, which means we're close to it. That means the drone has found the target, and as such can continue to its next target.

In any case, each column in the matrix is summed, such that we have an array of values. Each of these values is the total activation in a specific column of the heat map. A high activation at a certain position means it is likely that the target is in that position, and that means the drone should fly in that direction. To figure out the "centre of gravity" of the activation, we calculate the weighted average of the indexes of the columns, with the array of values as weights. The result can be normalised, giving us the direction the drone should fly.

3.2.3 Implementation

We used NumPy to deal with the matrix calculations.

The implementation has several constants that were abstracted from in the previous section.

The multiplication constant $c_{cooldown}$ was set to 0.5. In our experience, larger values caused "after images" to persist longer than needed, and smaller values reduced the positive effects of the heat map too much, making jitters and missing blobs too strong.

The downsize factor by which the matrix was scaled down from the drone camera was set to 4, meaning that each cell in the heat map represented 16 pixels. This was chosen because a smaller downsize factor would make the algorithm that activated the heat map too slow, and with a downsize factor of 4, the resolution preserved was still sufficient for subtle steering decisions.

The activation added for each blob depends on the blob area A. This was another way to reduce the effect of jitters. The formula that worked best for us was:

$$activation = 50 + \frac{\sqrt{A}}{10}$$

Here, *activation* is the value that is added to each element in the heat map matrix that fits inside the current blob. That operation is repeated for each blob the blob detection has found in the current frame.

Above formula was chosen because the size of the blob has an influence on the activation. Smaller blobs tend to be false positives, while larger blobs are more likely to be true positives. Because this property has more to do with the width and height of the blob than with the area of the blob, the square root of the area was taken (which is the geometric mean of the width and the height). A relatively large constant of 50 is added for each activation. We have no explanation for that, it simply worked better.

The threshold that decided whether the drone has seen *anything* interesting was set to 16, which is chosen fairly arbitrarily.

The threshold for deciding the target has been found and the drone was finished or needed to find another target, is strongly dependent on the real life size of the target. Our targets were about $\frac{1}{4}m^2$ and the constant that worked best was 30000.

3.2.4 Example

This is an example of how the activation is updated on the heat map. For this example, we use a much larger downsize factor for clarity, namely 80. We will pretend a large blob was seen some number of frames before, so not all values in the heat map are initially zero.

The heat map matrix looks like this before cooldown and before the new activation is added:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 20 & 20 & 20 \\ 0 & 20 & 20 & 20 \end{pmatrix}$$

First, cooldown is applied:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 \\ 0 & 10 & 10 & 10 \end{pmatrix}$$

A large blob is then found in the upper-left corner, and a small jitter found in the upper-right corner. The values that have to be updated are underscored below:

$$\begin{pmatrix} \underline{0} & \underline{0} & 0 & \underline{0} \\ \underline{0} & \underline{10} & 10 & 10 \\ 0 & 10 & 10 & 10 \end{pmatrix}$$

For each of the blobs, the activation value has to be calculated. We assume for this example that the actual dimensions of the blobs are multiples of the downsize factor (which in actuality is often not the case, but it simplifies the calculations in this example).

For the large blob:

$$A = 160 \times 160$$

$$\begin{aligned} activation &= 50 + \frac{\sqrt{160 \times 160}}{10} \\ &= 50 + \frac{160}{10} \\ &= 50 + 16 = 66 \end{aligned}$$

For the small blob:

$$A = 80 \times 80$$

$$\begin{aligned} activation &= 50 + \frac{\sqrt{80 \times 80}}{10} \\ &= 50 + \frac{80}{10} \\ &= 50 + 8 = 58 \end{aligned}$$

These values are added to the heat map:

$$\begin{pmatrix} 66 & 66 & 0 & 58 \\ 66 & 76 & 10 & 10 \\ 0 & 10 & 10 & 10 \end{pmatrix}$$

3.3 Control

3.3.1 Flying towards objects

```
direction = XvalueOfAverageOfActivation
turnTowards(direction)
flyForward
```

`XvalueOfAverageOfActivation` is determined by using this formula: (as explained in 3.2.2)

$$\texttt{XvalueOfAverageOfActivation} = 0.5 - \frac{Avg}{i_{max}}$$

$$Avg = weightedaverage([0, ..., i_{max} - 1], weights = weights)$$

$$weights_i = \sum_{j=0}^{j_{max}-1} H_{ij}$$

$$H = \text{(heat map matrix, } i_{max} \text{ columns, } j_{max} \text{ rows)}$$

Example With the heat map from 3.2.4, here is how to calculate `XvalueOfAverageOfActivation`:

$$H = \begin{pmatrix} 66 & 66 & 0 & 58 \\ 66 & 76 & 10 & 10 \\ 0 & 10 & 10 & 10 \end{pmatrix}$$

$$weights = \begin{pmatrix} 132 & 152 & 20 & 78 \end{pmatrix}$$

$$\begin{aligned} Avg &= weightedaverage([0,1,2,3], weights = [132,152,20,78]) \\ &= \frac{0 \times 132 + 1 \times 152 + 2 \times 20 + 3 \times 78}{132 + 152 + 20 + 78} \\ &= \frac{426}{382} \approx 1.12 \end{aligned}$$

$$\texttt{XvalueOfAverageOfActivation} = 0.5 - \frac{1.12}{4} \approx 0.22$$

A direction of 0.22 means steering moderately to the left.

3.3.2 Package inspection

```
if SumActivation > threshold
        currentTarget = getNewTarget(currentTarget)
    else
        startLanding
```

The sum of activation if determined by using this formula: (as explained in 3.2.2)

$$\texttt{SumActivation} = \sum_{i=0}^{i_{max}-1} \sum_{j=0}^{j_{max}-1} H_{ij}$$

$$H = (\text{heat map matrix, } i_{max} \text{ columns, } j_{max} \text{ rows})$$

Example With the heat map from 3.2.4, here is how to calculate `SumActivation`:

$$H = \begin{pmatrix} 66 & 66 & 0 & 58 \\ 66 & 76 & 10 & 10 \\ 0 & 10 & 10 & 10 \end{pmatrix}$$

$$\begin{aligned} \texttt{SumActivation} &= 66 + 66 + 0 + 66 + 78 + 10 + 0 + 10 + 10 + 58 + 10 + 10 \\ &= 384 \end{aligned}$$

4 Results

4.1 Github

The source code to the ROSMAV project (as well as this report) are accessible on Github. This is because it made working with multiple team members reasonably convenient. Additionally, it makes ROSMAV easily accessible to others, as they can simply clone or fork the repository from Github.

4.2 Tasks

In the end, the ROSMAV project consists of two tasks: following corridors and inspecting packets. The drone performs both of these tasks well, after proper calibration (or using pre-calibrated values if the colors have been calibrated in the same situation before). The drone has to be made to take off manually.

4.3 Setup

Setting this project up has proved to be difficult to streamline. We provide a virtual machine image that has already been set up to make it easier to use. In the appendix of this report a description has been included about how to set up your own workspace.

References

[1] International micro air vehicle conference and flight competition URL: *http://www.imav2011.org/*.

Appendix A

5 Background

5.1 IMAV

IMAV is a series of conferences and competitions with as main objective "to provide an effective and established forum for dissemination and demonstration of original and recent advances in MAV technology."[1] MAV stands for Micro Air Vehicles and IMAV stands for International Micro Air Vehicles.

5.2 BioMAV

The previous and first BioMAV project started in 2010. It was the entry of Radboud University Artificial Intelligence department. BioMAV was a large success and obtained the third price in the IMAV 2011 pylon challenge.

5.3 Biological inspiration

This project's biological inspirations are mostly related to vision. The corridor-following task that we first made was inspired by how flying insects such as moths follow the light of the moon for navigation. The final packet inspection task described in section evolved from animals hunting their prey.

In this section several of the implemented algorithms are discussed. Every subsection starts with the code in pseudo code and ends with an explanation of the algorithm.

6 Tutorial

During this project a lot of our dependencies have changed. This includes a new Ubuntu version, a new ROS version and our library for controlling the drone is deprecated. This makes it technically very difficult to reproduce our software in the original setting.

With some adjustments our software now works on newer versions of Ubuntu and ROS. It also uses a new library for the controlling of the drone. The complete project can be downloaded as an .ova file and can be imported into a virtual machine.

To use our software you can either use our virtual machine (which is recommended) or install it as a new Linux distribution. The instructions for installing our virtual machine can be found in section 6.1 and the instructions for installing a new distribution can be found in section 6.2.

6.1 Option a: Virtual machine (recommended)

Download virtual box at https://www.virtualbox.org/ and install this software. Download our virtual box image (currently available on:
`https://mega.co.nz/#!ONZR2DrY!fpG2KGBSurOtpoTNI323EhcyZscosjcvwL1S2Q55ZLw`) and add it to your virtual boxes. Possibly adjust your settings, it is known that not all settings work (especially the network settings seem to be specific to your hardware).

When controlling the drone, working settings for your virtual box are:
In network: enable the network adapter by checking the checkbox
In network: set attached to: bridged adapter

Note that the set attached to setting has to be changed to NAT when using "normal" internet again and that the bridged adapater is needed when controlling the drone.

Also note that the drone must be connected to wifi before launching your virtual box.

6.2 Option b: Installation

To install ROSMAV manually, follow the following steps:

1. Install Ubuntu (we used Ubuntu 10.10 and later 12.04).
2. Install ROS (we used Electric and later Fuerte) by following this guide:
   http://www.ros.org/wiki/electric/Installation/Ubuntu *or*
   http://www.ros.org/wiki/fuerte/Installation/Ubuntu
3. Make sure that you performed the "Environment setup" step during installation.
4. Download our repository from https://github.com/dutchcheesehead/ROSMAV
5. Navigate to ROSMAV.
6. Run **./install.sh**.
7. Close your terminal window.

6.3 Starting ROSMAV

6.3.1 Option a: using the new software (virtual box)

Before starting our software make the following preparations:

1. Prepare the drone by putting a charged battery into the drone, after this its lights will become red.
2. Connect to the drone by using wifi.
3. Start the ARDrone virtual box.

To start our software, run the following commands all in different terminal windows:

1. **roscore**
   This starts roscore, a possible result is visible in figure 4.

2. **rosrun ardrone_autonomy ardrone_driver**
   This starts the driver for the AR-Drone, a possible result is visible in figure 5.

3. **roslaunch cmvision blobs.launch**
   This starts the blob detection process, a possible result is visible in figure 6. The blue and red rectangles appear where the drone sees red and blue blobs.

4. **rosrun ardrone_tutorials keyboard_controller.py**
   This is needed to let the drone take off and land, the keys that are used for controlling the drone are shown in figure 7.

5. **rosrun image_view image_view image:=/heatmap_image**
   This allows you to see the heat map activation for the "inspect presents" task, and thus see through the eyes of the drone. A possible result of this is shown in figure 8. While our software is not running this window will be gray.

6. **rosrun ROSMAV inspectPresents.py** for the present inspection task *or*
   rosrun ROSMAV followLights.py for the task where the drone follows lights
   A result of this is visible in figure 9.

After all software is running click the image that is responsible for controlling the drone (the keyboard controller), go to the location where you want to fly (preferably with a lot of open space) and let the drone hover in the air (Please not that the spacebar can be used for an emergency landing). Our software will now take control of the drone.

Note that colours will appear different at different locations, see section 6.4 on how to change this.

Figure 4: Image of a terminal running roscore

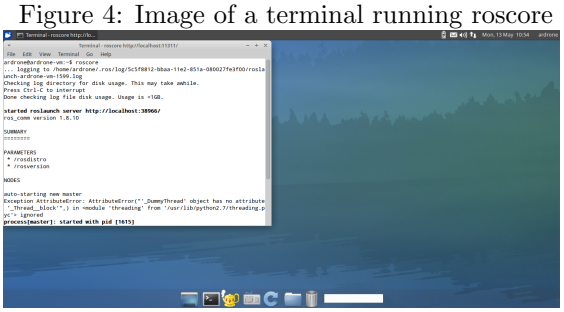


Figure 5: Image of a terminal running ardrone_autonomy

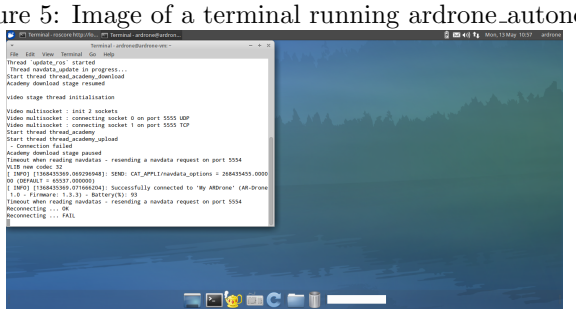


Figure 6: Image of a terminal running the blobslaunch script, note that an image will be displayed of what the drone sees with its camera

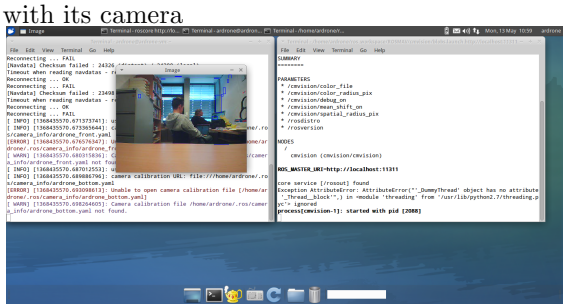


Figure 7: The keys needed for controlling the drone

```
PitchForward     = QtCore.Qt.Key.Key_E
PitchBackward    = QtCore.Qt.Key.Key_D
RollLeft         = QtCore.Qt.Key.Key_S
RollRight        = QtCore.Qt.Key.Key_F
YawLeft          = QtCore.Qt.Key.Key_W
YawRight         = QtCore.Qt.Key.Key_R
IncreaseAltitude = QtCore.Qt.Key.Key_Q
DecreaseAltitude = QtCore.Qt.Key.Key_A
Takeoff          = QtCore.Qt.Key.Key_Y
Land             = QtCore.Qt.Key.Key_H
Emergency        = QtCore.Qt.Key.Key_Space
```

Figure 8: Initial heatmap image

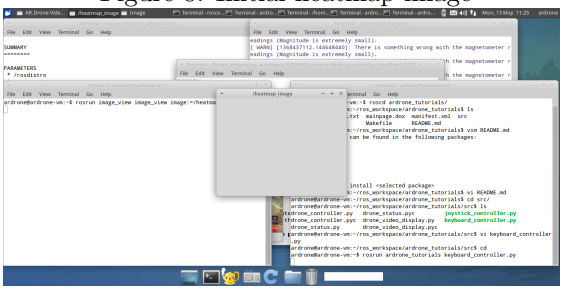
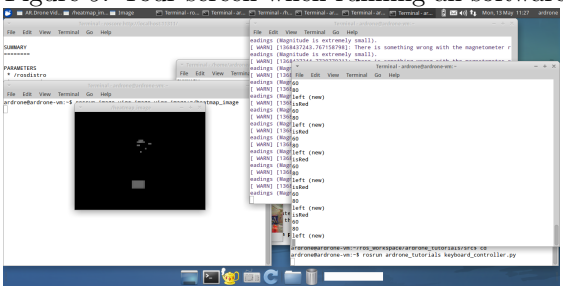


Figure 9: Your screen when running all software



6.3.2 Option b: using the old software

Before starting our software make the following preparations:

1. Prepare the drone by putting a charged battery into the drone, after this its lights will become red.
2. Connect to the drone by using wifi.
3. Start the ARDrone virtual box.

To start our software, run the following commands all in different terminal windows:

1. **roscore**
   This starts roscore, a possible result is visible in figure 4.
2. **rosrun ardrone_brown ardrone_driver**
   This starts the driver for the AR-Drone, a possible result is visible in figure 5.
3. **roslaunch cmvision blobs.launch**
   This starts the blob detection process, a possible result is visible in figure 6. The blue and red rectangles appear where the drone sees red and blue blobs.
4. **drone_teleop.py**
   This is needed to let the drone take off and land, the keys that are used for controlling the drone are shown in figure 7.
5. **rosrun image_view image_view image:=/heatmap_image**
   This allows you to see the heat map activation for the "inspect presents" task, and thus see through the eyes of the drone. A possible result of this is shown in figure 8. While our software is not running this window will be gray.
6. **rosrun ROSMAV inspectPresents.py** for the present inspection task *or*
   rosrun ROSMAV followLights.py for the task where the drone follows lights
   A result of this is visible in figure 9.

After all software is running click the image that is responsible for controlling the drone (the keyboard controller), go to the location where you want to fly (preferably with a lot of open space) and let the drone hover in the air (Please not that the spacebar can be used for an emergency landing). Our software will now take control of the drone.

Note that colours will appear different at different locations, see section 6.4 on how to change this.

6.4 Adding different colors to inspect

Suppose you want to add a different color to inspect.

1. Run **roscore**
2. In another terminal window, start the AR-Drone driver: **rosrun ardrone_brown ardrone_driver**
3. Start the ColorGUI: **rosrun cmvision colorgui image:=/ardrone/image_raw**
   You will see a window with the camera images from the AR-Drone.
4. Resize the window so you can see the text fields.
5. Keep the object you want to inspect in the view of the drone.
6. Click on the image of that object in the ColorGUI on a few different places.

7. Move both the object and the drone a bit around and click some more, to account for changes in lighting.
8. Repeat until you are confident it recognizes the object correctly in different circumstances.
9. If you made a mistake, close the ColorGUI and go back to step 3.
10. If it recognizes the object correctly, open "cmvision/colors.txt". Add both the color and the threshold. For the color, you can copy the last color and change the first bit into what the ColorGUI said, and change the name of the color. For the threshold, paste the threshold after the last threshold.
11. Edit "inspectPresents.py". You might need to add a function like "isRed" if it's not already in there. Then modify the "nextTarget" dictionary to the sequence you want to inspect the presents.

Figure 10: A picture of a drone.