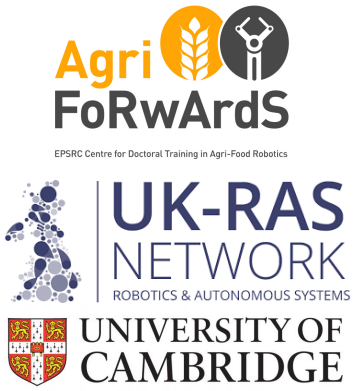
Agriforwards CDT and UK-RAS STAR Summer School 2021

Motor-Vision Coordination Kit

Grzegorz Sochacki
Fumiya Iida
Veronica Egorova



Tutorial Instruction

Motor-Vision Coordination Kit

Grzegorz Sochacki
Fumiya Iida
Veronica Egorova
University of Cambridge
Department of Engineering

April 2021

The following material was prepared for Agriforwrds and UK-RAS STAR Summer School, which took place in June 2021.

Table of Contents

List of Tables

List of Figures

1. Introduction

Fig. 1. Example robot chassis.

This is an instruction for a three day workshop where day one is used to assemble and test the hardware part of the robot. Second day will treat about exploring software and basic capabilities of the kit, while the last day is left open for own exploration through a quick science project.

This instruction details some items which you should prepare beforehand, as well as the whole assembly process. It will involve gluing with epoxy, wiring and assembly. The coding part involves programming Arduino and coding in Matlab. There are 4 tasks with solutions which give introduction to kit API.

1.1 Things you need to prepare

We would like to send you all required materials, but in some cases its not possible. Therefore there is a list of stuff you will need to to get in advance. These should be all cheap easy to get items, which usually can be found in every house. These things are:

- Old cloths - we will use epoxy, if it gets on your cloths there's no way to remove it.
- Few pairs of one-use gloves - protection for the epoxy staying on you hand for a week.
- Epoxy mixing - plastic plate and plastic knife could work, any cardboard should be ok too. Can even cut it out of plastic packaging of we will send you. Example of epoxy mixing can be found at https://www.youtube.com/watch?v=Ojsi2H-p21c (0:48). one of my setups during prototyping is shown in Figure 2.
- Laptop - 2 USB sockets will be needed - controlling camera and Arduino. Please have matlab installed - specific libraries will be installed during the session.
- MATLAB - it would save time if you had the matlab already installed as it may take time to do so, especially if getting a licence from the department is tricky.
- Internet connection - we will need to install some add-ons for matlab, so there will be some files to download.

2. Robot assembly and programming

The exercise is combination of work with software and hardware, therefore it's quite important to follow the order of assembly. In this way you can use time in which the epoxy is drying for installing software and programming. The proposed assembly order is:

- Glue the robot chassis.
- Install matlab and corresponding libraries
- Install Arduino IDE and corresponding libraries
- Wire up the system and test it
- Move the system into the chassis

2.1 Chassis Gluing

It's time to glue the chassis of the robot. We will use an epoxy glue for this task. The glue comes in a syringe with two chambers. In order to use the glue you need to squeeze it out on some surface like a piece of cardboard or one-use plate and mix the two substance together. The simplest setup for mixing can look something like in Fig 2. Single use gloves and old clothes are recommended as the epoxy takes quite a while to get of the skin. Since the moment the substance are mixed they start to harden, but there is no hurry - it will be at least 5 minute before it becomes an issue. You don't need to mix it longer than 1 minute for sure - just mix until the colour is the same across the load.

Fig. 2. A simple setup for epoxy mixing made out of used enveloped and part of a plastic package.

For now we will glue only the sides of the box, and the rotating plate. There are 7 pieces of acrylic in the package. There two large square-ish parts which form bottom and top parts of chassis, and four smaller rectangular pieces forming the walls of the box. We are now interested in large square part without a hole(bottom of the box) and the sides for the box. We want to glue the side to the bottom first - start with putting the components in the shape of the box without glue and think which areas should be covered with glue - then apply glue to these area like in the Figure 3 left. Now you can place first wall in its place - but mind that the epoxy will not harden for a while. Therefore, its recommended to place 2nd wall as soon as possible to have a stable construction as shown in Figure 4 left. When

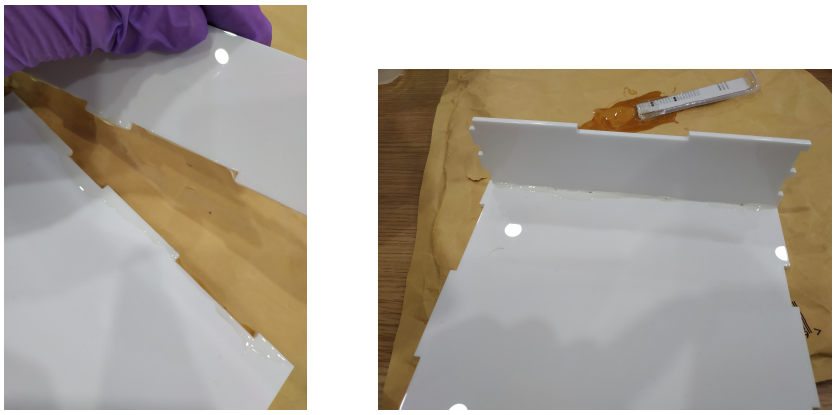


Fig. 3. Reasonable amount of epoxy applied to the box wall (left), first wall placed (right).

Fig. 4. Further stages of gluing the box.

adding 2nd and any next walls I recommend putting epoxy o all areas of the wall peace which will be in contact, but not adding glue to already placed parts as they are not settled yet. Finally, we should have something like in Fig 4 right. NOTE - DO NOT GLUE THE TOP YET, OR ACTUALLY NEVER.

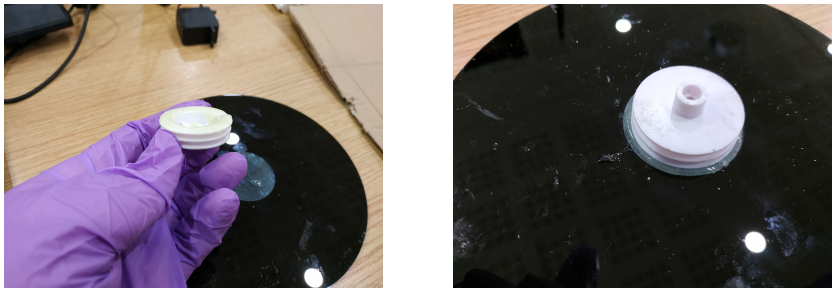


Fig. 5. Gluing the rotating plate to the servo, required parts(left) and glued part(right).

The next part is gluing the connection between the servo and rotating plate(acrylic circle). It could be glued to white 'disc' which comes with your servo, and we will only need its inner part. You need to apply glue only to the upper 'ring' area of his part as shown in Fig. 5 left. There is an engraved circle, which is slightly larger then the white part - it is

there to help you glue it in the middle of the plate. Glue the part in the middle of this circle and press down lightly. The expected effect is shown in Fig. 5 right.

2.2 Matlab installation

We assume that you are able to access Matlab through your university. The latter part of the instruction assumes a bare installation of the Matlab. At this stage you should download code which will be used for this workshop form github at https://github.com/Grzegorr/Robot-CDT-Summer-School. The folder Matlab code is our interest at the moment.

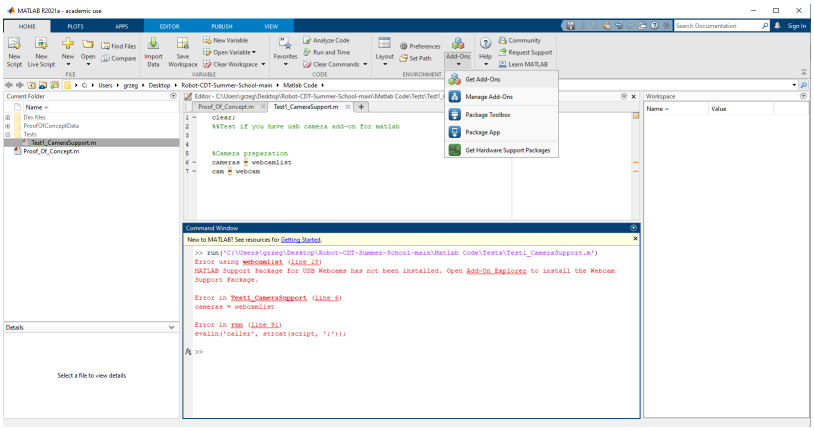


Fig. 6. Error message in case when there is no USB camera package installed.

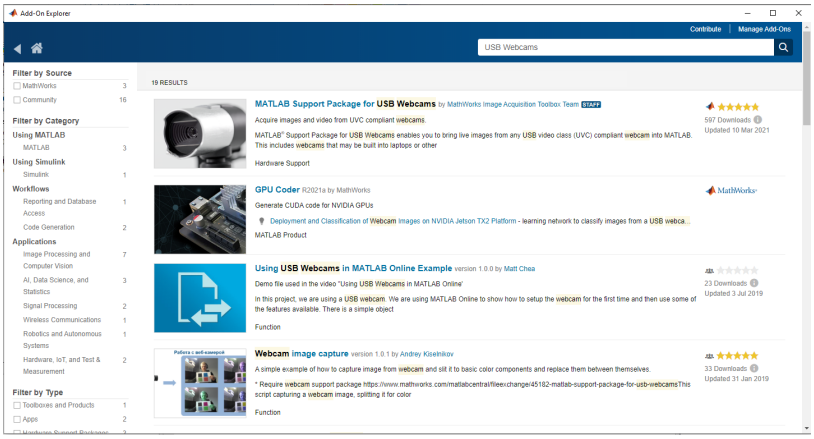


Fig. 7. The top search result is the correct package to install.

USB Cameras Add-On - This ad on is used to easily get access to USB cam from matlab and needs to be installed. To check if you have it installed go to Matlab Code/Tests and run 'Test1_CameraSupport.m'. If you get error message, like in Figure 6, follow prompts in the error message or go to 'HOME' tab and use Add-on drop down menu to install the correct add-on by searching for 'MATLAB Support Package for USB Webcams'. The correct package to download in a top result in the Figure 7. The last step is to run the test again to check if everything works, the correct outcome of the test is shown in Figure 8.

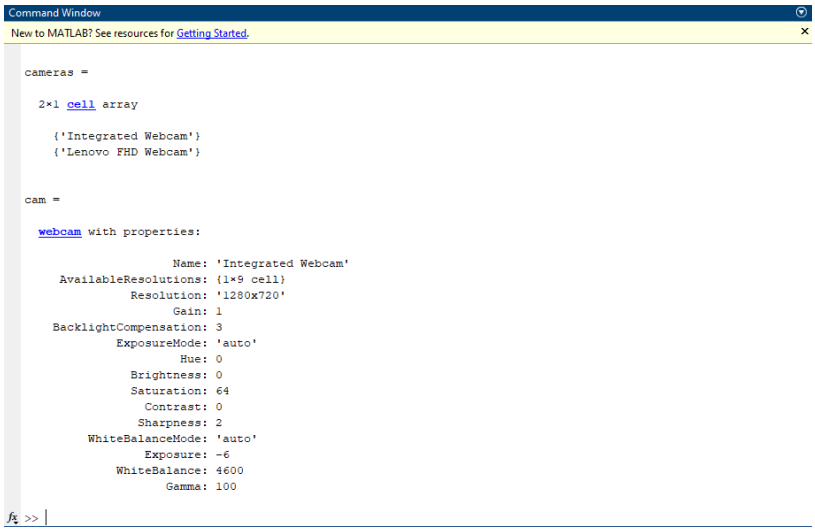


Fig. 8. Correct result of Test 1 - usb camera support.

2.3 Arduino IDE installation

Arduino IDE can be downloaded from 'https://www.arduino.cc/en/software', installation should be straightforward. It is a software which is used to code Arduino and transfer the code to microcontroller. This is a place where connecting the Arduino will be usefull. The setup for now should look just like in Figure 9. Now open Arduino IDE and load 'Robot-CDT-Summer-School/Arduino Code/USB_position_control/'. Try to load the code to the controller by arrow in top left corner of IDE window. In case of troubles, make sure if the board is set to 'UNO'. Also write down the USB port used by Arduino as it will be used later to speak to it during robot operation. The way to check the port is shown in Figure 10. Installation is successful when no errors are prompted in the command window of Arduino IDE.

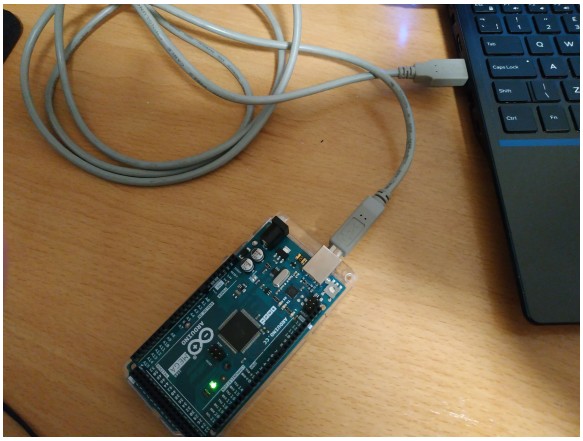


Fig. 9. Setup for Arduino IDE installation.

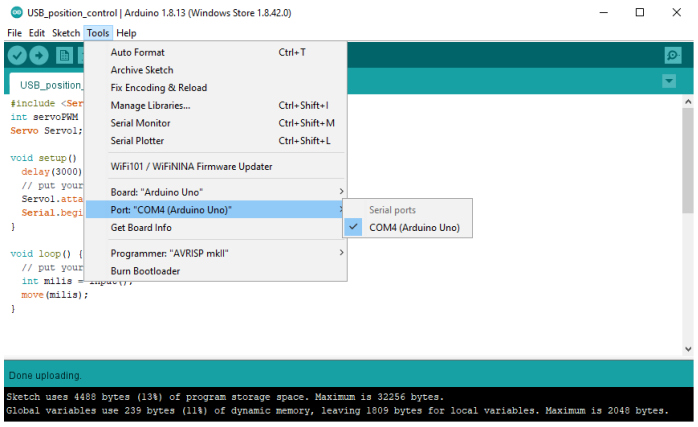


Fig. 10. Figure showing the way to check which USB port is used by Arduino.

2.4 Wiring The Circuit

It's time to wire our microcontroller and servo. Let's start by looking a little bit more at these components. Firstly, starting with a servo, it's shown in Figure 11. Servo in a package containing a motor, gearbox and controller. Usually it is controlled by a single signal, which is a goal position for the controller. After setting the goal position, the controller will take care of the motor control and move to the goal position. Looking back at the servo, it has 3 connections. Red and black are as always supply supply voltage and ground(+ and - of a battery). The last connection is a white wire, which takes the goal position signal. This signal is a 50Hz pulse-width-modulation(PWM) signal, hence with period of 20ms. The servos are usually controlled with signal pulse lengths between 1ms and 2ms, with 1.5ms being the central position as shown in Fig 12. There are libraries which help with generating this signal, and you will get a function in matlab to move servo to a comanded position, so you do not need to worry about it too much. It should also be said, that our servo is more specific and can do multiple rotations as it is designed for folding and unfolding sails.

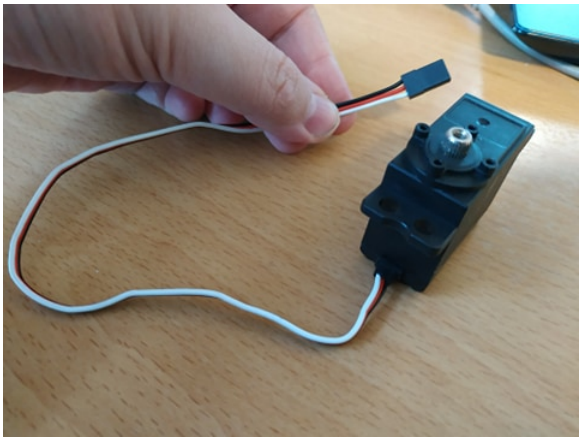


Fig. 11. Figure showing the servo to be used, with connector used by the servo.

As we need to generated the specific control signal, it is time to have a look at the Adruino pinout, as not all pins can produce such a signal. We will need to use one of the "PWM" pins for that measure. The explanation of all the pins is shown in Figure 13. Pin 10 is used by default in the arduino code provided. You can change it if you want, but the pin in code and in hardware must match.

Now we know enough to start wiring the circuit. Firstly, there should be a battery holder and arduino in your parcel. Take them out, but do not put the batteries in, the voltage of 4 AA batteries in series may damage arduino when not connected properly. Batteries should be put in at the end when you are sure that your connections are correct.

First thing that we need ensure while wiring is that both servo and arduino share a common ground. It means that their ground voltage is the same, which is not always the case. Physicists would use infinitely far away point as a ground reference, house installation may use actual ground below your house (which may be quite different then voltage in

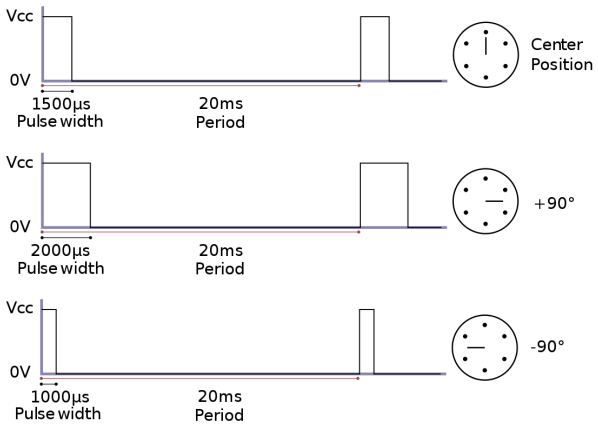


Fig. 12. Diagram showing control signal required by servomotor to move it to a specific position.

ground below transformer station on your street if loads are unbalanced),and if you power something from battery the ground voltage is a voltage of its ”-” end. The voltage is always a difference of potentials between two points, and when we give a single number for a voltage we need to remember the reference we use. This concept is very important for as as the arduino will send the PWM signal with reference to its ground, and the servo will read this same signal in reference to its ground, and for correct operation we want these grounds to be same voltage. We can achieve it by connecting these two grounds together. Otherwise the arduino’s ground voltage is determined by the computer’s USB port ground voltage, and the servo ground is determined by it’s supply’s ground voltage.

Anyway to get common ground connect ”-” of the battery to arduino’s GND pin, doesn’t matter which one. Now connect the servo’s ”-”(black wire) to another GND pin of the Arduino. As all the GND pins are connected on the board the servo’s ground, battery holder ground and Arduino ground are now connected and sit on the same voltage. Now connect the batteries ”+” to servo ”+” - red cable from battery holder to red cable at servo. The last connection is to be done with second jumper wire - connect pin 10 on Arduino to the white cable on the servo. The whole circuit should look like in Figure 14. Following the color codes are useful in long term, try to do that. Now you can put batteries in - it is likely that the servo will move at this point as it will get some undefined signal on whit wire.

FRIENDLY WARNING: All these ground pins I talked about are connected together and connected to the GND pin of your USB port. This pin is connected to motherboard of you PC. Therefore if you ever touch anything high voltage(9+ volts?) to any area connected to ground of this system you may fry your PC. And it can be as simple a jumper wire which slides out of connector, so be careful if you want to alter the circuit!

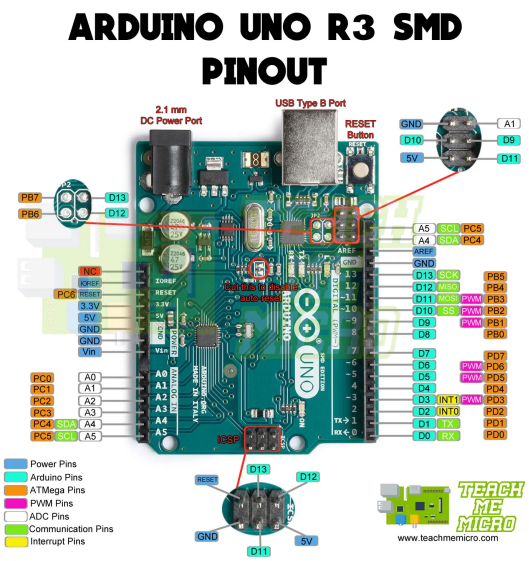


Fig. 13. Pinout of Arduino UNO, showing capabilities of various pins.

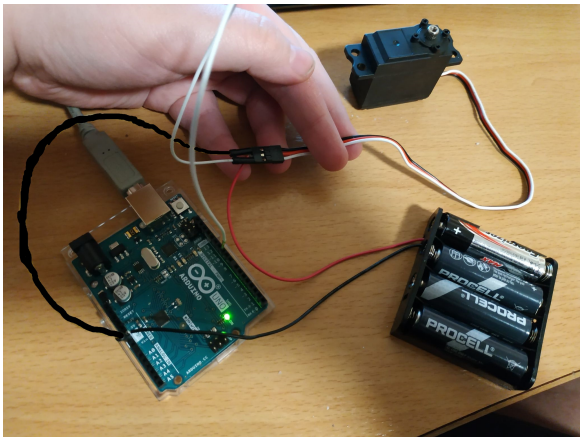


Fig. 14. Figure showing correctly wired system. Ground pins of both battery pack and servo connected to Arduino GND pin.

2.5 Finishing The Robot Assembly

Finally it is time for final assembly. You should be starting with the circuit wired in previous section and also bring the glued chassis. You should start with items as shown in Figure 15 left. Now its time to disconnect just few wires to make it easier to move the circuit into the chassis. You need to disconnect USB cable and the servo for the next part - it's easiest to do it at the servo connector as shown in Figure 15 right. Now put the servo into the hole of the top cover, letting the cables go first. Now it s time to connect back the USB cable. You will need to thread it through the hole (only type A connector will fit through) and connect back the servo. After these steps you should have something like in Figure 16 left. Now you can place the lid on, but don't glue it. It will be locked in place by design and you will always have an option to get inside to fix any issues. The robot should look like in Figure 16 right. Now attach the plate to the servo shaft and you should have an effect like shown in Figure 17. Now its time to glue our system with epoxy again - glue the battery holder and the Arduino inside the robot, also glue the servo to the lid. Now the robot is ready to operate.

Fig. 15. Early stages of the robot assembly, focusing on preparing the circuit for moving it inside the chassis. Circuit before required disconnections (left) and with required disconnection on the right.

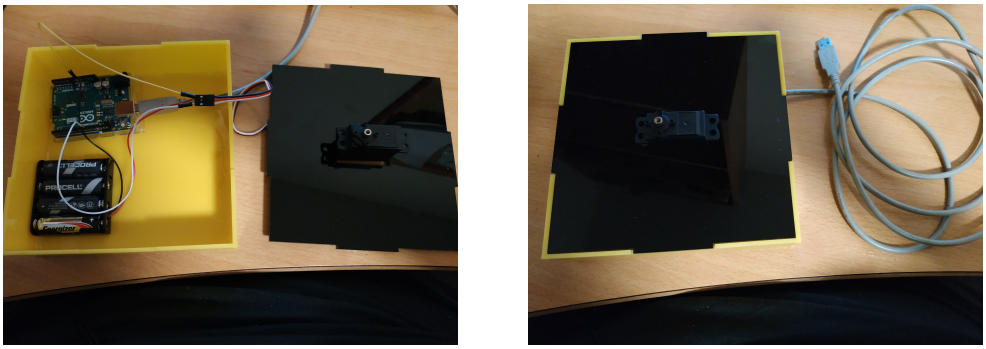


Fig. 16. Late stages of robot assembly. Connecting the wires again (left) and mounting the top part.

Fig. 17. Successfully assembled robot, with grapes as an example object for CV exercise.

3. Exercises

This section contains a list of exercises which will give you all the skill needed to play with the kit on your own. We will start with learning how to operate the servo from matlab as well as take and manage picture in subsection 3.1. This section also discusses troubleshooting the kit in case of any problems.

A deeper dive into servo control is taken in subsection 3.2, which discuses how to limit the servo speed from the matlab for large moves. It also discusses some edge cases of servo control.

3.1 Task 1. Move motor, take a picture.

This exercise is designed to show you basic functionalities of the kit. You will learn how to move the motor to a set position, take a picture, save it under a specified name, and store it for later processing. The code for this task is available at https://github.com/Grzegorr/Robot-CDT-Summer-School/tree/main/Matlab%20Code/Exercises/Task1 .

Lets open the Task 1 file and seen what is going on there. The code starts with these two commands

```
clear;
clc;
```

, which clear the command window and clear all stored variables. This is usually used just for convenience, but in this case it may be very important in some cases. **This is mainly because some of the variables in MATLAB will be objects responsible for connection to camera and USB port. This may cause other programs to not have a permission to connect to this specific USB. This is one of troubleshoots when you can't access your USB ports.**

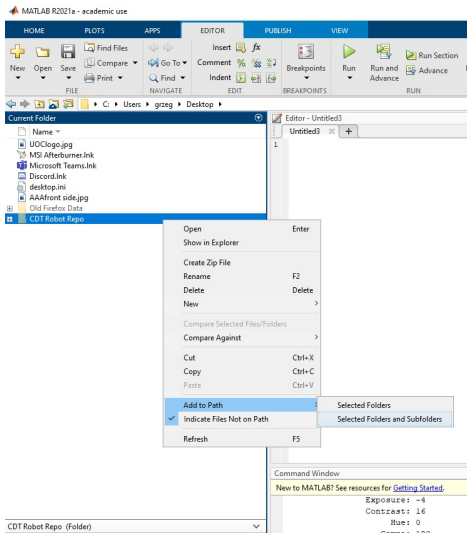


Fig. 18. Adding the repository to the MATLAB path, to get access to all functions prepared before workshop.

Moving Further into the code we encounter the following piece of code:

```
serialportlist
s = serialport("COM4",115200);
s.Terminator;
configureTerminator(s,"LF");
```

This code will list for you available serial ports connected to your PC. It will probably show no entries unless your Arduino is connected. Note the "COM4" in the second line - that's

the name of the port the matlab will speak to so **you may need to adjust that value.** The next two line will not be of our interest in this task.

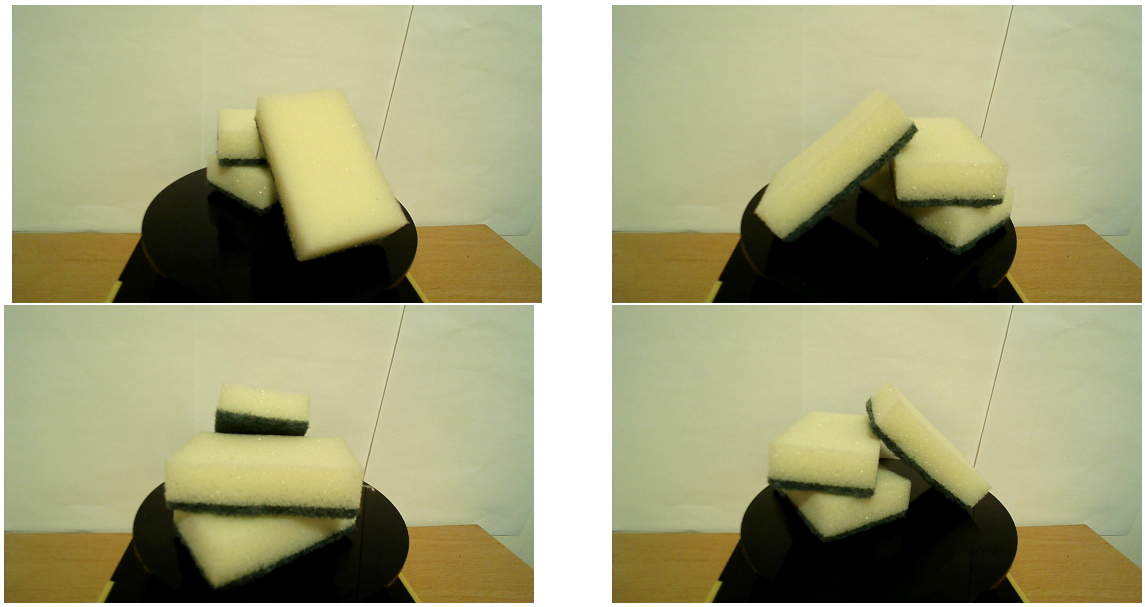


Fig. 19. Example solution of task 1 using sponges as observed item.

The next piece of code

```
cameras = webcamlist
cam = webcam
```

takes care of preparing the camera we use. The first line will list available cameras for you, and the second line will set the first available camera as a camera to use. Webcam should be used automatically, but you may use one of listed camera names to choose a specific camera by changing ”webcam” to ”webcam(2)”, ”webcam(3) etc.

Moving to the next line,

```
Angle_Move(s,0,5);
```

we see a custom function used, which we will use to move the servo. It takes 3 arguments, the first one ”s” is the serial port we specified before, no need to change it in at least first few exercises. The second argument is and angle from a servo starting position. It is in degrees, and the servo can handle +/-1200 degrees, but rather limit yourself for +/- 800 for better accuracy. The last argument is amount of time to pause the matlab for before executing next line of code - it is to give the servo time to move. **You will also need to include this script into the matlab path. It is best to do it by downloading the whole repo and the addin the whole repo into matlab path as shown in Figure 18**

Next two lines of code show how to take a picture.

```
img = snapshot(cam);
imshow(img);
```

first line is taking the picture and saving it in variable ”img”. The 2nd line is showing the taken picture in pop up window - good for testing, but comment it out if annoying.

Next lines are saving the picture.

```
file_name = "front side";
folderpath = "Pictures from sides/"
imwrite(img, folderpath + file_name + ".jpg");
```

This will save the picture ”img” in a folder of path ”folderpath” under name ”file_name”. **Mind that the path is counted from the folder you are currently in (look at the right hand side of matlab window it will show you the current folder).**

Now it’s time for you to expand on this code. How about making the robot to move the object you place on the plate, photographing it from four sides and saving the pictures in a folder? You can peak at Task1_Solved to see the answer. An example solution with a few sponges is shown in Figure 19. Try to also run the task twice - you are likely to get a large, fast movement at the end. It will get even worse if you would leave the servo far away from the starting position. But do not worry we will look at servo control with a little more depth in the next task!

3.2 Task 2. Deeper Look on the servo control.

We will investigate the reason for the large movements occurring in code from last exercise and see its origin. We will also see that it can be much, much worse if the function prepare before the workshop would not be programmed properly. So to kick off, we probably expect the movements to come from the fact the the servo comes back to it's middle position at the begining of the code execution. Then it could be countered by slowly moving the servo to the 0 degrees position at the end of our program. And we will do that for a start, but I promise there will be a deeper dive into servo control after this.

Firstly, we actually do not know the deflection of the servo for most of the time. In such a case the function to move the servo back to the middle position would need a user specified argument. We want to do better then this, therefore we will add the following line of code to the Angle_Move function:

```
last_position = angle;
```

This line will store for us the last command given to the Angle_Move, so we know by how much we need to move the servo back/forward.

The next task is write a simple for loop which moves the servo back to the starting position. It could be done with a following code:

```
no_moves = floor(last_position/10)
for 1:no_moves
    Angle_Move(s,last_position - 10, 2);
end
Angle_Move(0);
```

Analyzing it line by line: first we assume we will move back to the middle in steps of 10 degrees. Then we divide our position by 10 and take the floor of this value, so for example in case of position of 108 degrees we will get 10, as a floor(10.8). Then we move down by 10 degrees each step. We can use the notation used as the last position will be updated each time we call Angle_Move function. Add this code at the end of Task 1 code and check if that works. I also will leave you to adjust the code handle negative last position. If you can't do that the solution is available in Task2_Solved and the Return_To_Middle Preprepared function it uses.

Now the juicy bit I have promised. We will see that the nature of the servo control requires a little bit more care in the control. So far we just moved the servo to 0 degrees, and we are happy that it works. But how is the 0 degrees defined after all? We need to look at the Move_Angle script itself. Please have a look at that script and familiarize yourself with it. The following line specifies the relationship between the angle and sent signal:

```
%Starting point
us = 1500;
```

```
us = us + (angle / 2);
us_str = num2str(us);
write(s,us_str,"uint8");
```

The code start with specifying the signal for zero degrees, which is a pulse length of 1500 us. It is a middle of the servo range, as we discussed during the circuit construction - have a look at it again if you already forgot. Now change the starting point to some other value and run the Task2_Solved again. Our program does not prevent the jerky movements anymore. It seems like the servo gets a signal of 1500us at the start anyway and we actually cannot control it. So all our exercise was actually to bring that control signal slowly back to 1500us rather then to 0 degress.

Why is the signal 1500us set automatically at the program start? I do not know, maybe that's initialization of the "servo" instance in arduino, or maybe it's in the hardware? You may check it out as one of your projects. Also, we were storing the last angle in the matlab which is kind of a proxy, would be better to know the signal value from the arduino for sure. You may expand the the communication protocol to work both ways and get this value straight from the controller as you research project. Also don't forget to set it back to 1500 before you move to next tasks.

3.3 Task 3. Computer vision for future feedback

In this task we will use an AprilTag [1] to to get a visual feedback. We will stick it to our rotating plate and read it's orientation. It is useful feedback which we could use to get information about movement speed (no control of it with servo) or just check on accuracy of the movement. This could for example replace our wait time in MoveAngle() function with a tolerance on the pose(as we usually have with robotic arms), what could greatly increase the speed and flexibility of programming.

You have already some experience with the kit, therefore this task will not be explained step after step. Also, this task is based on MATLAB tutorial [2]. It has some extras but both tutorial and a skeleton for code I provide should be enough for anyone.

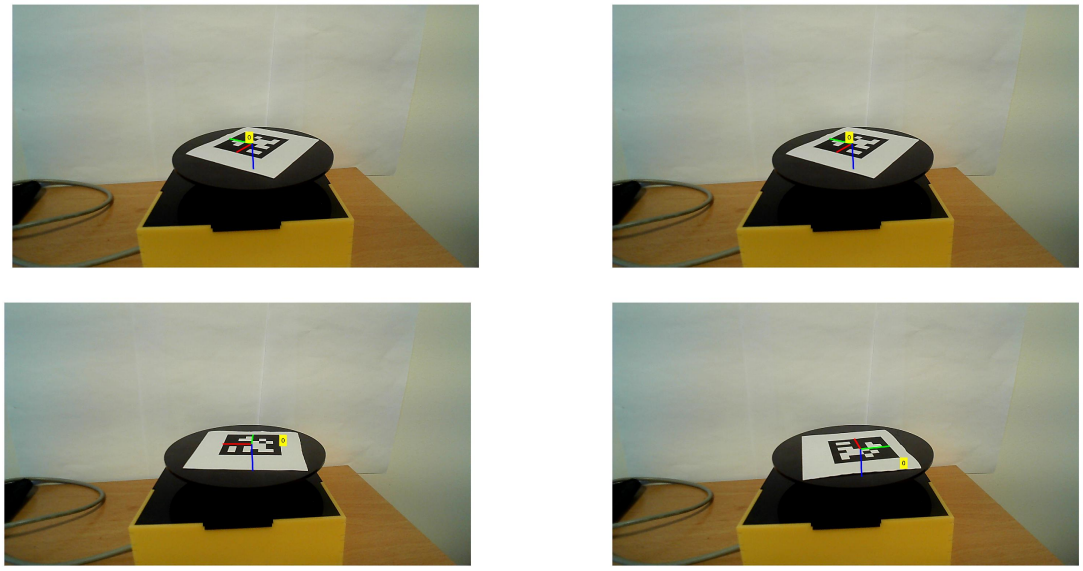


Fig. 20. Example solution of task 3, providing the visual feedback of the curent position.

I did the task before and few advice are:

- A lot is available in MATLAB packages like the whole detection and reading of the tag, as well as changing the rotation matrix to Euler angles.
- You will need to supply tag family. The family used is "tag36h11".
- Size of the tag is 6 cm, it is size of the black square.
- Don't cut the white area around the tag, it is needed.

The Expected result of the task is shown in Figure 20. In addition to that the rotation in radians should be identified. In the next task we will base the robot movements on the

3.4 Task 4. Visual servoing – skewed tags not permitted

In this task we will make sure the the robot adjusts the angle nicely in case we skew the tag. The solution to this task is available on the repo as previous tasks.

A variation of the code used in task 2 can be used. I would suggest while loop which measures the current rotation with the code we produced in task 3 and then moves servo by 10 degrees in correct direction. This will of course not settle well, so you will need to tweak it. For example make smaller steps while approaching the goal and increasing the settle time. The major challenge is to tackle the fact that we red the position instantly, while the hardware will actually have a settle time. There is a solution to this task provided, but you are statring from scratch.

4. Day 3 Research Ideas

This section provides a list of proposals for the quick research project which is to be done in day 3. Feel free to choose one or come up with your own idea.

4.1 Visual servoing without sticker (eg. Pen, scisors)

We used to get the the orientation vector from the AprilTag. But it more realistic and organic environment you cannot apply tags. So can you do this without a tag? How about using one of ”orange pens”. It should be possible to mask out the orange part of the pen, and its plug separately, find the direction between these and use it as direction? The question is how do you handle false positives? AprilTag could have just not been detected, and the loop can be just broken, but can we do that with just colour masking?

4.2 Extend communication with Arduino

Currently its only one way, but can you make arduino report back the last set position? Or can it detect if servo is working with a microphone? All of this would need expansion of communication protocol. Currently it's just send a number - set PWM length to this number of us. Wouldn't it be fun to make your own protocol from scratch?

4.3 Counting grapes

We are providing you with a bunch of fake grapes (not eatable!). We wonder if it's possible to segment and count the number of grapes. The grapes are very clamped and densely packed but the ability to rotate (and by a known angle!). We can rotate and take unlimited number of pictures, discard the ones we are unhappy with and retake them, so maybe this could help.

4.4 Stitching pictures

We have an automated way to take a picture from different sides. Can we make a ”reverse panorama” out of them? Maybe the Blender people can make an automated way to do digital models? Can we have these object in URDF so we can put them in rviz/pybullet?

4.5 Depth vision - can we use shadows?

There is an opportunity to place the source of light at known spot and investigate shadows while rotating. Can the general shape be estimated? Maybe a volume too? This one is tricky so start with very basic shapes is recommended.

4.6 Face tracking

Why to use the kit to just rotate the label if you could do some face tracking? The idea is to place the camera on the rotating plate, detect faces and move camera so the face is in the middle. That should cause the kit to track your face. Maybe run the Kalman filter form one of Lincoln courses?

4.7 Count Number of strawberries on the bush

That may be easy with the red ones, but can you do the green ones?

4.8 Seed separation and counting

We have a way of both observing and acting, therefore we could get a pile of mixed seeds, use spinning to separate them. This should let for an easier use of computer vision to for example estimate the proportion of the seeds.

Acknowledgments

We would like to thank CDT for funding the summer school.

References

[1] Olson E (2011) Apriltag: A robust and flexible visual fiducial system. *2011 IEEE International Conference on Robotics and Automation*, , pp 3400–3407. https://doi.org/10.1109/ICRA.2011.5979561

[2] Detect and estimate pose for apriltag in image - matlab readapriltag - mathworks united kingdom. Available at https://uk.mathworks.com/help/vision/ref/readapriltag.html.