Umeå Universitet

An Autonomous Robot for Detection and Management of Invasive Plants using a Convolutional Neural Network

Oliver Lindström
oliver.lindstrom00@gmail.com

Simon Sondén
simonsonden@gmail.com

September 24, 2019

Abstract

Invasive species are becoming increasingly prevalent in the natural fauna which in turn causes a disturbance in ecological systems. The management of these plants can often be a time consuming and difficult task due to the share number of plants. In this paper we propose an automated approach of management by building a robot which drives towards specific flowers after they have been identified using a convolutional neural network. This is then done by equipping a tracked robotic vehicle with a Raspberry Pi and a camera running an image recognition python script using a neural network which detects Dandelions. The result show a 85% accuracy of the neural network and that the preimage analysis algorithm is capable of finding dandelions in a field. The result however, also show that the robot only is able to find and drive towards a specific dandelion in a testing environment which did not work in a real meadow. This mixed result is discussed to be caused by the image analysis program having too broad of a definition for yellow and that the used camera's poor picture quality made it hard for the neural network to find the distinguishing characteristics of the different flowers. This makes us conclude that an automatic approach of removing invasive plants would likely be feasible with a higher resolution camera and small modifications to the software.

Acknowledgements

We would like to thank Sven Rönnbäck, Senior Lecturer at the Department of Applied Physics and Electronics, Umeå University, for helping us with this project. Without his help this paper would not have been possible. He has helped us by providing us with locale, equipment, and advice. He was also the person who applied for the necessary funding for this project. Thank you.

Contents

1 Introduction

1.1 Background and Project Goal

Invasive plants are a large problem in almost every country on earth. In Sweden there are many different invasive plants such as *Lysichiton americanus*, The American skunk cabbage and *Heracleum mantegazzianum*, Giant hogweed[1]. These plants are often difficult to kill as there are often a lot of them and as manual labour is often used to control the spread of them which can be expensive.

The goal of this project is to develop a robot prototype which can target and cut down a specific type of flower, the dandelion. Dandelions are not considered invasive in Sweden but they are very common and therefore they have been selected for this project.

To achieve this goal AI technology is used to identify dandelions in an image and kinematic steering equations are derived in order to construct a robot which can drive towards located dandelions. An example of dandelions is shown in figure 1.

Figure 1: Picture of a Dandelion

1.2 Hardware

In this project we will be using several kinds of hardware components to accomplish the task. These components will both be used for the movement of the robot but also for the navigation. A schematic of the circuit can be sen in figure 2 and all the hardware used is listed below:

- Raspberry Pi 3B
- 12 Volt batteries x3
- 24 Volt motors x2
- Arduino Uno
- DC-DC converter
- MD03 - 24 Volt 20 Amp H Bridge Motor Drive x2
- Chassi with tracks

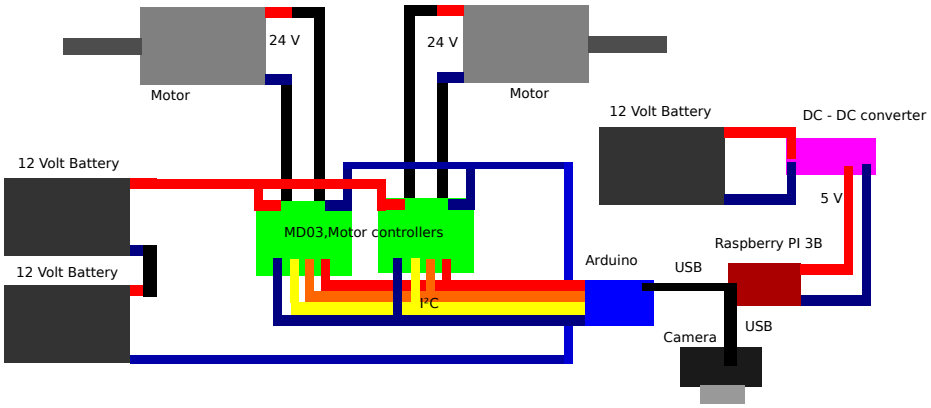

Figure 2: A schematic of the circuit and protocols used

1.2.1 Tiny Computers and Microcontrollers

A computer powerful enough to host the neural network while still being small enough to fit in the robot will be needed for this projekt. This report will be using a Raspberry Pi 3B[2]. The Pi features a powerful tiny processor capable of analyzing the images which will be fed to it by the camera. It will be running the Debian based operating system Raspbian.

This report will also use a Arduino Uno[3] as a link between the Raspberry Pi and the motor controllers. It will take input from the Pi using a serial connection and send it to the motor controllers using the protocol I^2C.

1.2.2 Motor controllers and batteries

The motor controllers, MD03[4], then receives the output from the Arduino and can adjust the angular velocity and acceleration of the motors. The controllers work by using a circuit of transistors called a h-bridge. This setup will allow a lower current to adjust the greater one coming from the batteries.

This report will also use two $12V$ lead–acid battery as the power source for the robot. Since some of the components such as the Raspberry Pi are rated for a lower input voltage, a DC-DC converter will be used to achieve these lower voltages.

1.3 Classified Data

In order to train a neural network a large amount of classified data are needed. A large amount of the data used in this paper has been classified specifically for this paper but some of the data was taken from a flower dataset from Oxford University [5].

1.4 GitHub

All code and all data used in this project can be found in the following GitHub repository: `https://github.com/Zigolox/DandelionFindingRobot`.

2 Purpose

In this section the purpose of each experiment or task conducted in order to build the robot will be described.

2.1 Correlation of Motor Torque, Speed and Voltage

The purpose of this experiment is to find a relation between the torque acting on a motor, the speed of the motor, and the output voltage of the motor.

2.2 Calculating Camera Angle from Pixel Location

The purpose of this experiment is to find a relation between the coordinate of a pixel on a camera image and the angle α to the object displayed in reality.

2.3 Preprocessing and Creation of Subimages

The purpose of this experiment is to construct a program to analyse the image and do the necessary preprocessing to analyse subimages in order to see if there are dandelions in them.

2.4 Classification of Flowers with a Neural network

The purpose of this experiment is to determine weather or not a part of an image contains a dandelion.

2.5 Construction of the Robot

The purpose of this experiment is to describe how the final robot is constructed using the results from all other experiments and sub tasks.

3 Theory

In this section the important theoretical concepts used to build this robot are discussed.

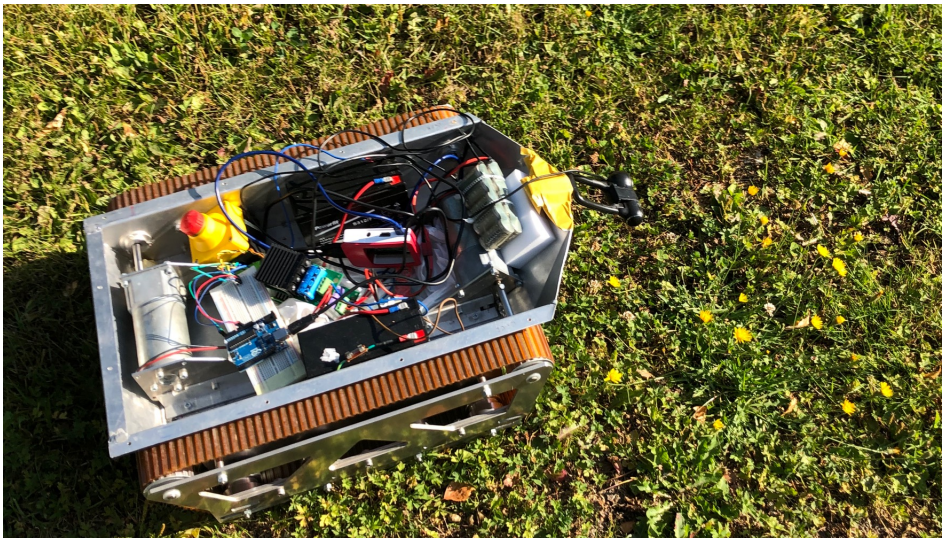

Figure 3: A picture of the robot

3.1 Equations for Steering on Surface with Slipping

Regular two wheel differential steering [6] is very easy to describe mathematically as both wheels can roll without slipping. This is not the case when four wheels that cannot rotate are mounted on a robot in any way. Therefore this section will show the important equations to describe the motion of a tracked vehicle as seen in figure 3.

$$v = \frac{v_1 + v_2}{2} \tag{1}$$

$$\omega = \frac{D}{D^2 + L^2}(v_2 - v_1) \tag{2}$$

Where:
v is the velocity of the robot
v_1 is the velocity of the inner motors in the rotation
v_2 is the velocity of the outer motors in the rotation
D is the width of the robot
L is the length of the robot
ω is the angular velocity of the robots rotation around its center

A full derivation of these equations can be found in the appendix. In the

appendix an equation for calculating the torque acting on each motor is also derived.

These equations are used, together with the results of section 6, to create a program which sets the velocity of the robot motors.

3.2 Calculating Camera Angle from Pixel Location

To find a relation between α and the pixel coordinate N the fact that for a right hand triangle $tan(\alpha) = D/L$, where D and L are the sides of the triangle, will be used.

3.3 Correlation of Motor Torque, Speed and Voltage

In order to create a steering algorithm it is necessary to know how a DC motor behaves when different voltages and torques are applied to it. A simple model to describe a DC motor can be found in [7]. This model can be simplified to the following equation.

$$u = A\omega + B(\tau_{ext} + \tau_{in}) \tag{3}$$

Where:
u is the applied voltage to the motor.
ω is the angular velocity of the motor.
τ_{ext} is the applied external torque.
τ_{in} is the constant internal torque.
A and B are constants to be determined experimentally.

To simplify calculations non SI-units are used in this report. The units are chosen such that the maximum applied voltage corresponds to 255 units and the maximum output velocity of the motors is 255 units. The quantity torque can be replaced by

$$\tau = \frac{Mg}{r} f_c. \tag{4}$$

Where:
M is the mass of the robot.
g is the standard gravity.
r is the radius of the wheels attached to the motors.

f_c is a dimensionless constant which will be referred to as the force constant throughout this report.

Using this quantity f_c Equation (3) can be simplified to

$$u = \alpha\omega + \beta f_c + u_0. \tag{5}$$

Where:
α, β, and u_0 are constants to be determined experimentally.

3.4 Preprocessing and Creation of Subimages

In this section OpenCV is used to read and analyse images [8]. OpenCV is a python module containing many different tools to analyse images which have all been heavily optimized.

3.5 Classification of Flowers with a Neural network

To determine if an image contains a dandelion a Convolutional Neural Network or a "CNN" is used [9][10]. For this particular robot the open source python module TensorFlow Keras, has been used to create this CNN [11][12]. The reader is assumed to be familiar with the basic theory behind neural networks [13] and convolutional neural networks [14]. If not the citations of this paper, especially those in this section, give an adequate introduction to the topic.

3.6 Object Location

Using only one camera to determine the distance to an arbitrary object is difficult. However if the position of the object is expressed in spherical coordinates [15] the two angles θ and ϕ can be found. These angles can be found by finding a relation between the pixel coordinate (N, M) relative to the center of the image and these angles.

To find the third coordinate, the distance to the object, R the following method can be used. If an object of some characteristic length L_0 , in this case any length proportional to the square root of its area on the screen,

located at a distance R_0 from the screen is moved to a position at distance R its new characteristic length L will obey the relation.

$$\frac{R_0}{L_0} = \frac{R}{L} \tag{6}$$

Thus if the approximate characteristic length of the average dandelion L_0 at some distance R_0 is known the distance to dandelions can be found with relatively good accuracy.

4 Materials

In this section the materials used to conduct the experiments in this paper will be presented.

4.1 Calculating Camera Angle from Pixel Location

The materials used for this experiment are

- Camera
- Ruler

4.2 Correlation of Motor Torque, Speed and Voltage

The materials used for this experiment are

- Robot
- Plane with the ability to be inclined
- Protractor

5 Method

5.1 Calculating Camera Angle from Pixel Location

This experiment was conducted in the following way.

- Place a ruler at a distance D from the camera oriented such that the ruler is orthogonal to the optical axis of the camera.
- Use the markings on the ruler to find a relationship between the distance from the center of the image in pixels N and the corresponding real world distance from the center on the ruler L.
- The relation between α and N can be found using $tan(\alpha) = \frac{L}{D}$.

5.2 Correlation of Motor Torque, Speed and Voltage

The experiment to determine α was conducted in the following way:

- Elevate the robot in such a way that its tracks can rotate but the robot cannot move.
- Measure some multiple of the period of rotations of the motors at different input voltages.
- Calculate the velocity of the robot measured in some unit ranging from 0-255 by multiplying the inverted period of each measurement with some conversion factor.
- Plot the voltage as a function of velocity.
- Perform a linear regression on the data and measure the slope α.

To determine β and u_0 a similar experiment was conducted in the following way:

- Place the robot on a plane that can be inclined.
- Measure the minimal voltage required for the robot to move at different inclinations θ.
- Plot the voltage as a function of $sin(\theta)$.
- Perform a linear regression on the data.
- Since $sin(\theta)$ is the force coefficient here β and u_0 can be measured from the plot.

5.3 Preprocessing and Creation of Subimages

The first step of analyzing the image is to find all yellow pixels in the image. This can be done using OpenCV. First the image is converted to HSV format [16] to determine which pixels are yellow. Then a mask [16] with all yellow pixels is created.

```
#Convert image to hsv format
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

#Lowest yellow hsv value
lowYellow = np.array([26,155,25])

#Highest yellow hsv value
highYellow=np.array([30,255,255])

#Binary image where all yellow pixels have value 1
#and the rest have value 0
mask = cv2.inRange(hsv, lowYellow, highYellow)
```

After this the contours of the yellow regions are found using the mask.

```
#Find contours in the binary image
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

These contours are used to find the area of all yellow regions in the image as well the radius and centre of the smallest possible enclosing circle for each contour.

```
areas = []      #List of the areas of all contours
centres = []    #List of the centers of all enclosing circles
radii = []      #List of the radii of all enclosing circles

#Loop through all contours
for c in contours:
    #Calculate area of the contour
    area = cv2.contourArea(c)

    #Find the centre and radius of the smallest enclosing #circle
    centre, rad = cv2.minEnclosingCircle(c)
```

```
        areas.append(area)
        centres.append(centre)
        radii.append(rad)
```

If the area of the contour is larger then a certain arbitrary predetermined smallest value a subimage containing the entire enclosing circle with a small margin is created and resized to match the input shape for the neural network.

```
    #Shape of image
    shape = img.shape[:2]

    #Left side of subimage
    left = int(max(origo[0].05*rad,0))

    #Right side of subimage
    right = int(min(origo[0].05*rad, shape[0]))

    #Top of subimage
    top = int(max(origo[1].05*rad,0))

    #Bottom of subimage
    bottom = int(min(origo[1].05*rad, shape[1]))

    #Create subimage
    subimg = img[top: bottom, left:right]

    #Resize subimage
    subimg = cv2.resize(subimg, (256,256), interpolation = cv2.INTER_AREA)
```

5.4 Classification of Flowers with a Neural network

The first part of building this Neural Network is to import the necessary modules and functions.

```
    # TensorFlow and tf.keras
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import regularizers
```

```
from tensorflow.keras.layers import Dropout, Flatten, Activation, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.applications import VGG16

# Helper library
import numpy as np
```

After this is done the marked data is read and split into test and train data.

```
#The data is read using a predefined function readData
all_data = read_data(feature_path,label_path)

#The data is split up into test and train data
train_features, train_labels, test_features, test_labels = all_data
```

When the data is read the model needs to be defined before it can be trained using the data. The first part of the model is the pretrained model VGG16 [17], with the top layer removed and with all layers except for the last 2 set to be untrainable.

```
#Read the VGG16 model without the top layer
vgg_conv = VGG16(weights='imagenet',
include_top = False,
input_shape=(256, 256, 3))

#Set all layers except for the last two to be untrainable
for layer in vgg_conv.layers[:-2]:
    layer.trainable = False
```

Now the model can be built and compiled using the VGG16 model together with a few extra layers. The model is built to three output classes, "Dandelion", "Buttercup", and "Other"

```
#Set up the model
model = keras.Sequential([
    vgg_conv,
```

```
    Dropout(0.5),

    Flatten(),

    Dense(32, activation = tf.nn.relu,
                kernel_initializer='random_uniform',
                bias_initializer='random_uniform',
                kernel_regularizer=regularizers.l2(1e-8),
                bias_regularizer=regularizers.l2(1e-8)),

    Dropout(0.50),

    Dense(32, activation = tf.nn.relu,
                kernel_initializer='random_uniform',
                bias_initializer='random_uniform',
                kernel_regularizer=regularizers.l2(1e-8),
                bias_regularizer=regularizers.l2(1e-8)),

    Dropout(0.50),

    Dense(3,activation = tf.nn.softmax,
                kernel_initializer='random_uniform',
                bias_initializer='random_uniform',)
])

#Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Figure 4 shows the structure of the neural network.

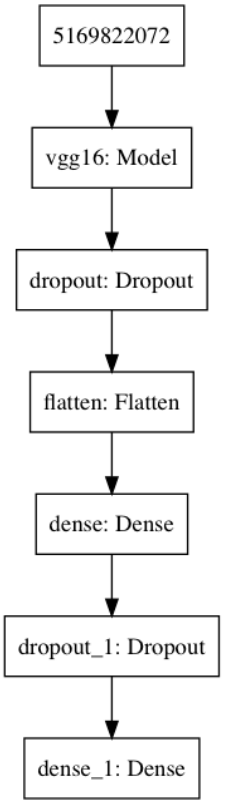

Figure 4: Structure of the Neural Network

After the model has been compiled it is trained with a validation split of twenty percent and the loss and the accuracy of the train and validation data are saved. The model is saved every time the validation accuracy increases. After the model has been trained the model is tested using the test data.

```
#Checkpoint (save model when validation accuracy increases)
filepath="model_{epoch:02d}_{val_acc:.3f}.h5"
checkpoint = ModelCheckpoint(filepath,
monitor='val_acc',
verbose=1,
save_best_only=True,
mode='max')
callbacks_list = [checkpoint]

#Fit model (50 epochs), 20% of all data is used as validation data
```

```
#All data is saved in history
history = model.fit(train_features,
train_labels,
validation_split=0.20,
epochs=50,
callbacks=callbacks_list)

#Test the model
test_loss, test_acc = model.evaluate(test_features, test_labels)
```

The data saved in the variable history is then plotted and test loss and test accuracy are printed.

5.5 Construction of the Robot

The final robot is constructed in the following way. Two 12 volt batteries are connected in series to two motor controllers which are connected to the motors of the robot. The Raspberry Pi is connected to the battery with the ground port of the system through a DC to DC converter. Then the Arduino Uno is connected to the Raspberry Pi through a serial connection and to the motor controllers using the SDA and SCL ports. The Raspberry Pi is also connected to a camera.

The Raspberry Pi processes data from the camera and sends instructions on which direction it should turn to the Arduino. The Arduino then recieves this data and uses it to calculate and set the speed of each motor.

6 Results

6.1 Calculating Camera Angle from Pixel Location

The results of the measurements are shown in figure 5.

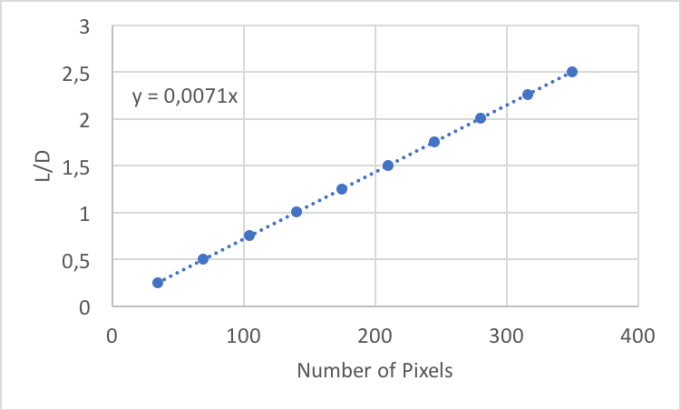

Figure 5: Plot showing $\frac{L}{D}$ as a function of Pixels

6.2 Correlation of Motor Torque, Speed and Voltage

The results of the two parts of the experiment are shown in figure 6 och figure 7.

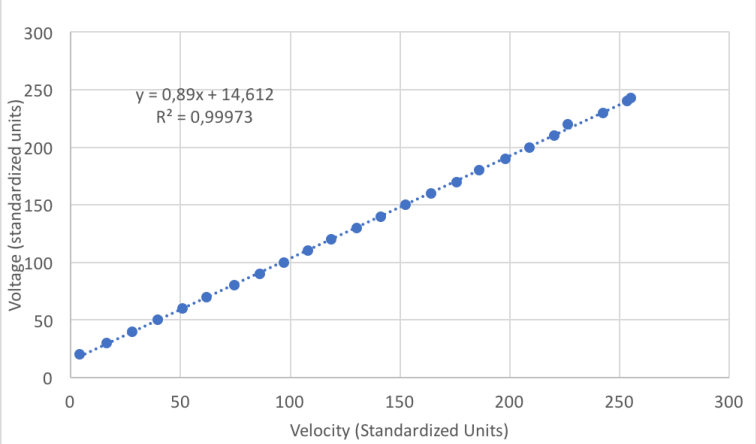

Figure 6: Voltage input as a function of velocity

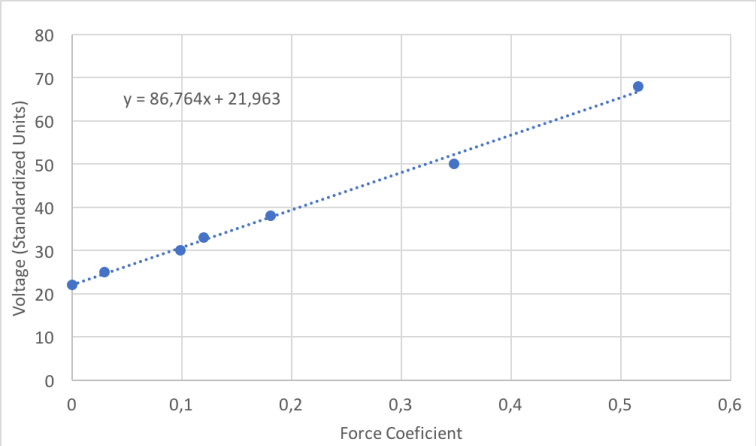

Figure 7: Voltage input as a function of force coefficient

From these two plots the values of the three parameters α =..., β =...,and u_0 =... can be read.

6.3 Preprocessing and Creation of Subimages

Figure 8 shows an example of regions being identified as yellow with their surrounding circles shown.

Figure 8: Picture of dandelions with yellow regions identified and marked

6.4 Classification of Flowers with a Neural network

The test accuracy for the model is: 85% The graph in figure 9 shows the accuracy of the model as a function of number of epochs.

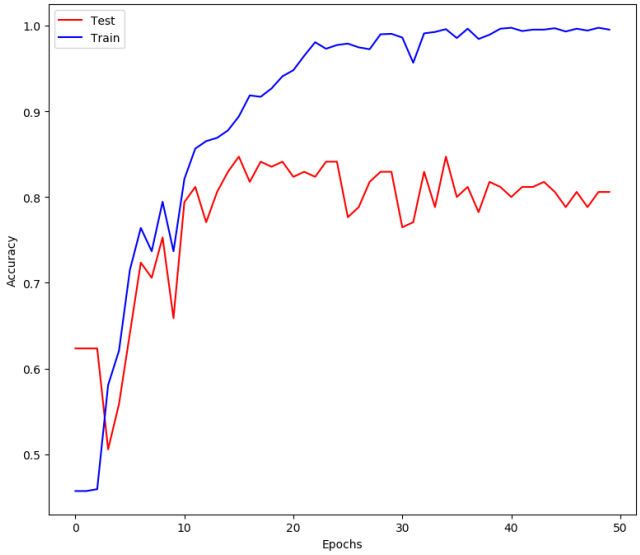

Figure 9: Plot of neural network accuracy

And this figure 10 shows the the loss of the model as a function of number of epochs.

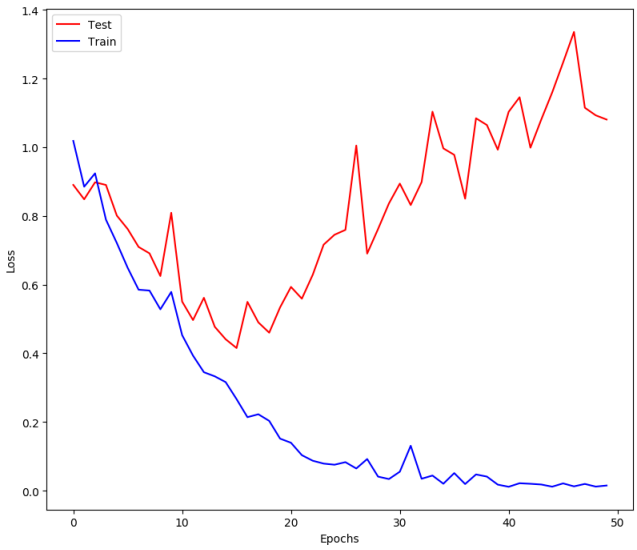

Figure 10: Plot of neural network loss

6.5 Final Robot

The results presented in this section are qualitative rather than quantitative.

The robot was able to show which regions of an image were yellow and which of the yellow regions contained dandelions with great accuracy.

The robot was able to steer towards a located item under ideal conditions where the item was very large.

The robot did not manage to steer towards dandelions on a real meadow under any conditions tested.

7 Discussion

In this section the results of this paper are discussed.

7.1 Camera Angle from Pixel Location

The results show a linear relationship between $tan(\alpha)$ and the pixel coordinate. The theory explaining why this is so will not be discussed in this paper but it should be noted that this seems to be the case.

7.2 Preprocessing and Creation of Subimages

As figure 8 shows some regions which do not seem yellow are considered yellow by the program. If a more narrow range of yellow is chosen however some regions that are part of dandelions might not be considered yellow. This problem might be fixable by setting a different range of yellow colors or by changing the image format from HSV [16] to something else but these results are good enough for this paper.

7.3 Neural network

The accuracy of the neural network, 85%, is not ideal. The quite poor results of the model can be explained by two main reasons. The first reason is that the train data was very limited in numbers and many images used as train data were also poor in quality. This could easily be fixed by spending more time classifying images of better quality but this was beyond the time scope of this paper. The second reason is that the computer used lacked a dedicated GPU [18] which means that training a large neural network would take to much time and therefore the project was limited by the hardware used to train the network.

7.4 Final Robot

The results of the final robot show that it was entirely possible to locate dandelions in an image and they also show that it is possible to drive towards located items. The failure of the robot when it was tested on an actual meadow can be explained by the fact that the resolution of the camera was not good enough to distinguish the dandelions from other regions considered yellow by the image analysis software. Another reason could have been that the there were different lighting conditions which made it hard for the pre-analysis software to distinguish the yellow in the picture.

8 Further Research

As this robot built did not preform very well when trying to find flowers in a meadow the next obvious step to research this subject further is to test if a better camera enables the robot to complete this task. Further research could also include constructing a robot which can locate different types of plants and avoid certain plants while targeting others.

9 Conclusion

While the final results of the project show that the robot could not drive towards dandelions it still managed to show that such a robot could be built if a camera of higher resolution is used. Therefore this report arrives in the conclusion that using the method of this report to construct a robot which targets dandelions or other plants is possible but it might require a camera with somewhat higher resolution than the one used in this report and may also need a filter to protect it lens from direct sunlight to mitigate the problems with different lighting conditions.

References

[1] Inkeri Ahonen. *Invasiva främmande arter – artfakta*. URL: https://www.naturvardsverket.se/Sa-mar-miljon/Vaxter-och-djur/Frammande-arter/Invasiva-frammande-arter/. [Visited 2019-09-14].

[2] *Buy a Raspberry Pi 3 Model B – Raspberry Pi*. Raspberry Pi Foundation, United Kingdom. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Visited 2019-09-14].

[3] *Arduino Uno Specification*. Wikipedia. URL: https://en.wikipedia.org/wiki/Arduino_Uno. [Visited 2019-09-14].

[4] *MD03 - 24Volt 20Amp H Bridge Motor Drive*. Robot Electronics. URL: https://www.robot-electronics.co.uk/htm/md03tech.htm. [Visited 2019-09-14].

[5] Maria-Elena Nilsback and Andrew Zisserman. *British flower datasets*. Feb. 2009. URL: http://www.robots.ox.ac.uk/~vgg/data/flowers/. [Visited 2019-09-14].

[6] Thomas Hellström. *Kinematics equations for differential drive and articulated steering*. Department of Computing Science, Umeå University, 2011-08-28.

[7] *DC-motor modelling and parameter dentification*. Lindköping University, Sweden. Apr. 2019. URL: http://www.control.isy.liu.se/student/lab/minseg/dcmotor/file/pm_dcmotor.pdf. [Visited 2019-09-14].

[8] *OpenCV*. Wikipedia. URL: https://en.wikipedia.org/wiki/OpenCV. [Visited 2019-09-14].

[9] Adit Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks*. URL: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/. [Visited 2019-09-14].

[10] *An intuitive guide to Convolutional Neural Networks*. Free Code Camp, Machine Learning. Feb. 2018. URL: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/. [Visited 2019-09-14].

[11] *Guide for Keras in TensorFlow Core*. Tenserflow, Google. URL: https://www.tensorflow.org/guide/keras. [Visited 2019-09-14].

[12] iamkhader. *2D Convolution Network in Keras/Tensorflow*. Kaggle, Digit Recognizer. URL: https://www.kaggle.com/iamkhader/2d-convolution-network-in-keras-tensorflow. [Visited 2019-09-14].

[13] Tony Yiu. *Understanding Neural Networks*. Towards Data Science, Medium. URL: https://towardsdatascience.com/understanding-neural-networks-19020b758230. [Visited 2019-09-14].

[14] *How Convolutional Layers Work*. Keras Documentation, Google. URL: https://keras.io/layers/convolutional/. [Visited 2019-09-14].

[15] Bronštejn Ilâ Nikolaêvič. *Handbook of mathematics*. [Visited 2019-09-14]. Springer, 2007. ISBN: 978-3-540-72122-2.

[16] Sadekar Kaustubh. *Invisibility Cloak using Color Detection and Segmentation with OpenCV*. Learn OpenCV. Feb. 2019. URL: https://www.learnopencv.com/invisibility-cloak-using-color-detection-and-segmentation-with-opencv/. [Visited 2019-09-14].

[17] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[18] *MacBook Air (13-inch, Early 2014) - Technical Specifications*. Apple. URL: https://support.apple.com/kb/sp700?locale=en_US. [Visited 2019-09-14].

[19] C. Nordling and J. Österman. *Physics Handbook for Science and Engineering*. Professional Publishing House, 2006. ISBN: 9789144044538. URL: https://books.google.se/books?id=xBAWGQAACAAJ.

A Derivation of Steering Equations

Consider a robot with velocity $\vec{v}$ rotating around its Center of Mass with some angular velocity $\vec{\omega} = \begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix}$. This robot has some motor i located at some point $\vec{r_i} = R(\theta) \begin{bmatrix} x_i \\ y_i \end{bmatrix}$ relative to the Center of Mass of the robot moving forward at some velocity $\vec{v_i} = R(\theta) \begin{bmatrix} 0 \\ v_i \end{bmatrix}$ where $R(\theta)$ is the 2D rotation matrix for rotations of an angle theta [15]. The velocity of the ground relative to the point on the motor touching the ground will be the direction for any force of friction acting on the motor. This velocity is called $\vec{\Delta v_i} = \vec{v_i} - \vec{v} - \vec{\omega} \times \vec{r_i}$. Plugging in all values this becomes:

$$\vec{\Delta v_i} = \begin{bmatrix} -v_x - v_i sin(\theta) + \omega x_i sin(\theta) + \omega y_i cos(\theta) \\ -v_y + v_i cos(\theta) - \omega x_i cos(\theta) + \omega y_i sin(\theta) \end{bmatrix}. \tag{7}$$

For simplicity, define $u_i = v_i - \omega x_i$. Furthermore let $\vec{v} = R(\theta') \begin{bmatrix} 0 \\ v \end{bmatrix}$. In the final resulting motion $\Delta\theta = \theta' - \theta$ will be very small and thus $sin(\theta') \approx sin(\theta) + cos(\theta)\Delta\theta$ and $cos(\theta') \approx cos(\theta) - sin(\theta)\Delta\theta$. This gives

$$\vec{\Delta v_i} = \begin{bmatrix} (v - v_i) sin(\theta) + (\omega y_i + v\Delta\theta) cos(\theta) \\ (v_i - v) cos(\theta) + (\omega y_i + v\Delta\theta) sin(\theta) \end{bmatrix}, \tag{8}$$

The length of this vector approximated to the linear order of $\Delta\theta$ is

$$\Delta v_i = q_i + \frac{\omega y_i v \Delta\theta}{q_i}, \tag{9}$$

Where:
$q_i = \sqrt{(v - v_i)^2 + (\omega y_i)^2}$

Since the force of friction is independent of the magnitude of velocity but parallel to the direction of the velocity [19] it is known that

$$\vec{F} = k\frac{\vec{\Delta v_i}}{\Delta v_i} = k(\frac{1}{q_i} - \frac{\omega y_i v \Delta\theta}{q_i^3})\vec{\Delta v_i}. \tag{10}$$

Thus the force acting on the robot can be calculated using:

$$\vec{F} = \sum_i (\frac{1}{q_i} - \frac{\omega y_i v \Delta\theta}{q_i^3})\vec{\Delta v_i}), \tag{11}$$

and there will be an equilibrium when this expression is 0. Note that k here is $k = \frac{mg\mu}{N}$ where N is the amount of motors if the weight of the robot is distributed equally over all motors.

On a four wheel robot there are be four different motors. These motors are located at positions $\vec{r_i} = \begin{bmatrix} \pm\frac{D}{2} \\ \pm\frac{L}{2} \end{bmatrix}$ and their velocities are such that the velocities of the motors on the left side of the robot are equal ($v_1 = v_3$) and those on the right side are equal ($v_2 = v_4$). Thus $u_1 = u_3 = v_1 + \frac{\omega D}{2}$ and $u_2 = u_4 = v_2 - \frac{\omega D}{2}$. Now let $v = \frac{v_1+v_2}{2} + \Delta v$ ($= \frac{u_1+u_2}{2} + \Delta v$). The variable Δv will be assumed to be small and therefore every expression will be approximated to the linear order of Δv. It will prove useful to define $q = \sqrt{(\frac{u_2-u_1}{2})^2 + (\frac{\omega L}{2})^2}$. This results in

$$\frac{1}{q_1} = \frac{1}{q_3} = \frac{1}{q} - \frac{(u_2 - u_1)\Delta v}{2q^3}, \tag{12}$$

$$\frac{1}{q_2} = \frac{1}{q_4} = \frac{1}{q} + \frac{(u_2 - u_1)\Delta v}{2q^3}. \tag{13}$$

Plugging this into Equation (11) and simplifying the expression results in the expression

$$\vec{F} = \frac{k}{q}(4(1-\frac{1}{4}(\frac{u_2 - u_1}{q})^2 \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix} \Delta v + 4(1-\frac{1}{4}(\frac{\omega L}{q})^2) \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} v\Delta\theta), \tag{14}$$

Since

$$\dot{\vec{v}} = \begin{bmatrix} -sin(\theta) \\ cos(\theta) \end{bmatrix} \dot{v} + \begin{bmatrix} -cos(\theta) \\ -sin(\theta) \end{bmatrix} v(\omega + \dot{\Delta\theta}), \tag{15}$$

there will be a stable force equilibrium at $\Delta v = 0$ and $\Delta\theta = -\frac{\omega}{\beta}$ where $\beta = \frac{1}{qm}k(2 - \frac{1}{2}(\frac{\omega L}{q})^2)$. The fraction $\frac{\omega}{\beta}$ is assumed to be very small and thus $\Delta\theta \approx 0$ as vll. Thus

$$v = \frac{v_1 + v_2}{2}, \tag{16}$$

$$\theta = \theta' \tag{17}$$

For the following calculations it is assumed that $\Delta\theta = 0$ and $\Delta v = 0$. Under

these conditions the torque acting on the robot can be calculated using the equation

$$\vec{\tau} = \sum_i \vec{r_i} \times \vec{F_i} \tag{18}$$

Only the scalar τ will be of importance as the vector will always be pointing in the z-direction. Plugging in the values for $\vec{F_i}$ using $q_i = q$ the torque can be found to be

$$\tau = \frac{k}{q} \sum_i x_i(u_i - v) - y_i^2 \omega \tag{19}$$

or, after simplifying.

$$\tau = \frac{k}{q}(D(u_2 - u_1) - L^2\omega) \tag{20}$$

This expression is 0 when

$$\omega = \frac{D}{D^2 + L^2}(v_2 - v_1), \tag{21}$$

when $\omega = \frac{D}{D^2+L^2}(v_2 - v_1) + \Delta\omega$, τ becomes

$$\tau = -\frac{k}{q}(D^2 + L^2)\Delta\omega, \tag{22}$$

meaning that this value for omega is stable.

For a bandwagon type robot the bands can be considered as an endless array of wheels infinitely close to each other. Thus Equations (14) and (20) multiplied by $\frac{dy}{L}$ can be used to calculate $d\vec{F}$ and $d\tau$. Thus

$$\vec{F} = \frac{1}{L}\int_{-L/2}^{L/2} \frac{k}{q}(4(1-\frac{1}{4}(\frac{u_2 - u_1}{q})^2 \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix} \Delta v + 4(1-\frac{1}{4}(\frac{\omega y}{q})^2) \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} v\Delta\theta)dy \tag{23}$$

$$\tau = \frac{1}{L}\int_{-L/2}^{L/2} \frac{k}{q}(D(u_2 - u_1) - y^2\omega)dy \tag{24}$$

This gives the result

$$v = \frac{v_1 + v_2}{2}, \tag{25}$$

$$\Delta\theta = 0, \tag{26}$$

$$\omega = \frac{D}{D^2 + \frac{L^2}{3}}(v_2 - v_1) \tag{27}$$

Finally the force acting on each track can be calculated by integrals in the same way as F was calculated. The calculations are skipped here but the result is shown here:

$$\vec{F_1} = \frac{u_2 - u_1}{4q} \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix}, \tag{28}$$

$$\vec{F_2} = -\frac{u_2 - u_1}{4q} \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix}, \tag{29}$$

Where:
F_1 is the force acting on the left track.
F_2 is the force acting on the right track.

When $v_1 = v_2$ (or rather when $\omega = 0$, $F_1 = 0$ and $F_2 = 0$. When $\omega \neq 0$ both forces will be in the direction of heading.

$$F_1 = mg\mu \frac{L}{2\sqrt{L^2 + (3D)^2}} sgn(\omega) \tag{30}$$

$$F_2 = -mg\mu \frac{L}{2\sqrt{L^2 + (3D)^2}} sgn(\omega) \tag{31}$$

Where:
m is the mass of the robot
μ is the coefficient of friction