**David Mühlenweg 201823392**

**Dokumentation Turtlesim Sprachsteuerung**

Benötigte Libraries:

**(Python Version 2.7.17)**

**PyAudio**

**gTTS(Google Text to Speech)**

**SpeechRecognition**

**ROS-Pakete**

**$pip install pyaudio**

**(je nach Linux version:** **$sudo apt-get install python-pyaudio python3-pyaudio)**

*https://pypi.org/project/PyAudio/*

**$pip install gTTS**

[*https://pypi.org/project/gTTS/*](https://pypi.org/project/gTTS/)

**$pip install SpeechRecognition**

[*https://pypi.org/project/SpeechRecognition/*](https://pypi.org/project/SpeechRecognition/)

**Generelle Funktion**

Das Programm startet ein Menü. Dieses Menü lässt sich wie alle anderen Funktionen über die Sprache steuern. Durch klares Aussprechen der jeweiligen Menüpunkte, werden diese geöffnet und ausgeführt. **Roscore** und **Turtlesim** werden automatisch gestartet.

1. **Zu Koordinaten fahren**
2. **In Richtung fahren**
3. **Steuerung**
4. **Beenden**

**Zu Koordinaten fahren:**

In dem ersten Menüpunkt, werden nacheinander die gewünschten Koordinaten abgefragt, die die Turtlesim ansteuern soll. Hier wurde das Beispielprogramm aus der Vorlesung erweitert (move2goal).

**In Richtung fahren:**

Indem zweiten Menüpunkt, kann die Turtlesim durch verschiedene Kommandos angesteuert werden. Mit Begriffen wie: vorne, links, rechts, hinten, schneller, langsamer, stopp usw. Alle Begriffe sind in der Menüansicht aufgeschrieben. Nachdem man die Turtlesim gestoppt hat, ist es des Weiteren möglich mit „letzte Strecke“, die letzte Strecke erneut zu fahren. Im Anschluss gibt es zudem noch die Möglichkeit, die gefahren Strecke in einem Bild (.png) abzuspeichern.

**Steuerung:** Gibt eine Übersicht über die Steuerungsmöglichkeiten an

**Beenden:** Schließt die Anwendung



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Monitor, Screenshot, Bildschirm, Telefon enthält.

Automatisch generierte Beschreibung

**Das Programm beinhalten folgende Programmstrukturen, die den Grundstein des Programms bildeten. Der Rest ist Eigenleistung und kann gerne weiterverwendet werden.**

**Guessing Game:** Beispiel Anwendung zu der SpeechRecognition Library [*https://realpython.com/python-speech-recognition/*](https://realpython.com/python-speech-recognition/)

**Move2Goal:** Beispiel Anwendung zur Ansteuerung der Turtlesim

[*http://wiki.ros.org/turtlesim/Tutorials/Go%20to%20Goal*](http://wiki.ros.org/turtlesim/Tutorials/Go%20to%20Goal)

**Besonderheiten im Programm**

**Nutzung von Threading:**

Das Programm benutzt Threading, was so viel bedeutet wie das „parallele“ ablaufen von Prozessen, dieses ist in dem Programm notwendig, da gleichzeitig Spracheingaben verarbeitet werden müssen und des Weiteren die Turtlesim in ihrer Position aktualisiert werden muss. Ein Linearer Ablauf ist hier nicht möglich, da der Loop sonst ständig unterbrechen würde, bis eine Spracheingabe des Users erfolgt. Des Weiteren muss das Menü dauerhaft angesteuert werden, um Befehle entgegenzunehmen.

Eine einfache Erklärung liefert folgender Link:

[*https://praxistipps.chip.de/python-threading-und-threads-so-funktionierts\_95975*](https://praxistipps.chip.de/python-threading-und-threads-so-funktionierts_95975)

Eine etwas komplexere Einführung gibt es hier:

[*https://www.python-kurs.eu/threads.php*](https://www.python-kurs.eu/threads.php)bzw. [*https://docs.python.org/3/library/threading.html*](https://docs.python.org/3/library/threading.html)

**Bugs**

Es gibt einen Bug in der SpeechRecognition Library, der die Zahlenwerte 8, 9 und 11 falsch darstellt. Diese werden als 8 Uhr, 9 Uhr und 11 Uhr ausgegeben. Im Quellcode wurde dieser Bug umgangen, damit eine normale Nutzung von allen Zahlenwerten möglich ist. Die entsprechenden Stellen, sind mit Kommentaren versehen

**Quellcode mit Kommentaren**

#!/usr/bin/python  
# -\*- coding: utf-8 -\*-

# ---------------------------#  
#        ERO-Projekt         #  
#     David Muehlenweg       #  
#         201823392          #  
# ---------------------------#

import sys  
import threading  
from PyQt5.QtCore import Qt  
from PyQt5.QtGui import QPixmap, QPainter, QBrush, QPen, QImage  
from PyQt5.QtWidgets import (QPushButton, QVBoxLayout,  
                            QApplication, QLabel, QMainWindow, QStyle, QWidget)

import rospy  
from geometry\_msgs.msg import Twist  
from turtlesim.msg import Pose  
from math import pow, atan2, sqrt

from gtts import gTTS  
import os  
import time  
import speech\_recognition as sr

# Liste von globalen Variablen die Thread übergreifen  
# aufrufbar sein müssen  
linea = 0  
angula = 0  
positionx = 0  
positiony = 0  
ActivateRefresh = True  
SaveData = False  
startPainting = False  
LineaList = [0]  
AngulaList = [0]  
xList = [0]  
yList = [0]  
AnzahlLoops = 0

class TurtleBotClass():  
    goal = Pose()  
    global recognize\_speech\_from\_mic

    def \_\_init\_\_(self):

        # Erstellt einen Knoten mit dem Namen 'turtlebot\_controller'  
        # und stellt sicher, dass es sich um einen  
        # eindeutigen Knoten handelt (mit anonym = True).  
        rospy.init\_node('turtlebot\_controller', anonymous=True)  
        # Publisher welcher auf dem Topic '/ turtle1 / cmd\_vel' veröffentlicht wird.  
        self.velocity\_publisher = rospy.Publisher(  
            '/turtle1/cmd\_vel', Twist, queue\_size=10)  
        # Ein Subscriber mit dem Topic '/turtle1/pose'. self.update\_pose wird aufgerufen  
        # wenn eine Nachricht von dem Typ Pose erhalten wird  
        self.pose\_subscriber = rospy.Subscriber(  
            '/turtle1/pose', Pose, self.update\_pose)  
        # globale Variablen instanzieren  
        self.pose = Pose()  
        self.rate = rospy.Rate(10)

    def update\_pose(self, data):  
        # Callback funktion, welche aufgerufen wird, wenn eine neue Nachricht vom typ 'Pose'  
        # vom Subscriber erhalten wurde  
        self.pose = data  
        self.pose.x = round(self.pose.x, 4)  
        self.pose.y = round(self.pose.y, 4)

    def euclidean\_distance(self, goal\_pose):  
        # Abstand zwischen Pose und Ziel  
        return sqrt(pow((goal\_pose.x - self.pose.x), 2) + pow(  
            (goal\_pose.y - self.pose.y), 2))

    def linear\_vel(self, goal\_pose, constant=1.5):  
        return constant \* self.euclidean\_distance(goal\_pose)

    def steering\_angle(self, goal\_pose):  
        return atan2(goal\_pose.y - self.pose.y, goal\_pose.x - self.pose.x)

    def angular\_vel(self, goal\_pose, constant=6):  
        return constant \* (self.steering\_angle(goal\_pose) - self.pose.theta)

    def getGoalFromUser(self):  
        goal\_pose = Pose()  
        startword\_is\_correct = False

        Startword = "starten"

        Trys = 1  
        MaxTrys = 5

        # Mikrofon und Recognizer instanzieren  
        recognizer = sr.Recognizer()  
        microphone = sr.Microphone()

        tts = gTTS(text="Um das Programm zu aktivieren, bestaetige mit starten", lang='de')  
        tts.save("audio1.mp3")  
        os.system("mpg321 audio1.mp3")

        # Spielraum fuer erneute Eingaben, falls etwas nicht funktioniert hat  
        for i in range(Trys):  
            for j in range(MaxTrys):  
                calledWord = recognize\_speech\_from\_mic(recognizer, microphone)  
                strCalledWord = str("{}".format(calledWord["transcription"]))  
                if calledWord["transcription"]:  
                    break  
                if not calledWord["success"]:  
                    break  
                print("Ich konnte Sie nicht verstehen. Bitte erneute Eingabe\n")

            # wenn es einen Error gibt, neu instanzieren  
            if calledWord["error"]:  
                print("ERROR: {}".format(calledWord["error"]))  
                break

            # Ausgeben des Strings in der Konsole  
            print("You said: {}".format(calledWord["transcription"]))

            # Wenn das eingegebene Wort gleich dem Startwort ist, Programm ausführen  
            if strCalledWord == Startword:  
                startword\_is\_correct = True

            if startword\_is\_correct:  
                print("Correct! You win!".format(Startword))  
                tts = gTTS(text="Bitte gebe die gewuenschte X Koordinate an", lang='de')  
                tts.save("audio2.mp3")  
                os.system("mpg321 audio2.mp3")  
                gotxcord = True  
                while gotxcord:  
                    xcord = recognize\_speech\_from\_mic(recognizer, microphone)  
                    zeichenx = str("{}".format(xcord["transcription"]))  
                    # Wenn zeichen numerisch ist und keine 'Bugzahl' ist, wird Koordinate gespeichert (umwandeln von String in Int)  
                    if zeichenx.isdigit() and zeichenx != "8 Uhr" and zeichenx != "11 Uhr":  
                        zahlx = int(zeichenx)  
                        goal\_pose.x = zahlx  
                        gotxcord = False  
                    # Ausgleichen eines Buggs, die zahlen 8 und 11 werden jeweils als 8 und 11 Uhr ausgegeben, dies wird hiermit uebergangen  
                    elif zeichenx == "8 Uhr":  
                        goal\_pose.x = 8  
                        gotxcord = False  
                    # Ausgleichen eines Buggs, die zahlen 8 und 11 werden jeweils als 8 und 11 Uhr ausgegeben, dies wird hiermit uebergangen  
                    elif zeichenx == "11 Uhr":  
                        goal\_pose.x = 11  
                        gotxcord = False  
                    else:  
                        tts = gTTS(text="Die Eingabe ist ungueltig", lang='de')  
                        tts.save("audio5.mp3")  
                        os.system("mpg321 audio5.mp3")

                print("Waehle die Y. Koordinate".format(Startword))  
                tts = gTTS(text="Bitte gebe die gewuenschte Y Koordinate an", lang='de')  
                tts.save("audio3.mp3")  
                os.system("mpg321 audio3.mp3")  
                gotycord = True  
                while gotycord:  
                    ycord = recognize\_speech\_from\_mic(recognizer, microphone)  
                    zeicheny = str("{}".format(ycord["transcription"]))  
                    if zeicheny.isdigit() and zeicheny != "8 Uhr" and zeicheny != "11 Uhr":  
                        zahly = int(zeicheny)  
                        goal\_pose.y = zahly  
                        gotycord = False  
                    elif zeicheny == "8 Uhr":  
                        goal\_pose.y = 8  
                        gotycord = False  
                    elif zeicheny == "11 Uhr":  
                        goal\_pose.y = 11  
                        gotycord = False  
                    else:  
                        os.system("mpg321 audio5.mp3")  
            else:  
                self.getGoalFromUser()  
            return goal\_pose

    def recognize\_speech\_from\_mic(recognizer, microphone):  
        # Uebertragen der Sprache von dem Mikrofon, gibt ein Dictionary zurück  
        # "success": Boolscher Wert ob API Anfrage erfolgreich war  
        # "error":   `None` wenn kein Fehler auftrat, ansonsten ein String mit einer  
        # Fehlermeldung, dass die API nicht erreicht werden konnte oder  
        # die Sprachaufnahme fehlgeschlagen ist  
        # transcription 'None' wenn Text nicht umgewandelt werden konnte, ansonsten  
        # String mit dem umgewandelten Text

        # Ueberprüfen, ob Erkennungs- und Mikrofonargumente vom richtigen Typ sind  
        if not isinstance(recognizer, sr.Recognizer):  
            raise TypeError("`recognizer` must be `Recognizer` instance")

        if not isinstance(microphone, sr.Microphone):  
            raise TypeError("`microphone` must be `Microphone` instance")  
        # Anpassung Erkennungsempfindlichkeit an Umgebungsgeräusche und zeichne Audio auf  
        with microphone as source:  
            recognizer.adjust\_for\_ambient\_noise(source)  
            audio = recognizer.listen(source)

        # Erstellen Antwort Objekt  
        response = {  
            "success": True,  
            "error": None,  
            "transcription": None  
        }

        # Versuchen, die Sprache in der Aufnahme zu erkennen  
        # wenn ein RequestError oder UnknownValueError abgefangen wird,  
        # wird das Antwort Objekt Aktualisiert  
        try:  
            response["transcription"] = recognizer.recognize\_google(audio, language ='de-DE')  
        except sr.RequestError:  
            # API konnte nicht erreicht werden  
            response["success"] = False  
            response["error"] = "API unavailable"  
        except sr.UnknownValueError:  
            # Sprache war unverständlich  
            response["error"] = "Unable to recognize speech"

        return response

    def move2goal(self):  
        # Faehrt die Turtle an die angegebene Position  
        distance\_tolerance = 0.1  
        # fuer die Funktion lokale Objekte instanzieren => kein self notwendig  
        vel\_msg = Twist()  
        # Debug Info  
        rospy.loginfo("Start Pose is %s %s", self.pose.x, self.pose.y)  
        rospy.loginfo("Goal is       %s %s", self.goal.x, self.goal.y)  
        rospy.loginfo("Distannce to Goal is  %f ", self.euclidean\_distance(self.goal))  
        rospy.loginfo("SteeringAngle to Goal is  %f ", self.steering\_angle(self.goal))

        while self.euclidean\_distance(self.goal) >= distance\_tolerance:  
            # Porportional controller

            # Lineare Geschwindigkeit in der x-achse.  
            vel\_msg.linear.x = self.linear\_vel(self.goal)  
            vel\_msg.linear.y = 0  
            vel\_msg.linear.z = 0

            # Winkel Geschwindigkeit in der z-achse.  
            vel\_msg.angular.x = 0  
            vel\_msg.angular.y = 0  
            vel\_msg.angular.z = self.angular\_vel(self.goal)

            # Publishing der Geschwindigkeit (velocity)  
            self.velocity\_publisher.publish(vel\_msg)

            # Publishen mit der festgelegte Rate  
            self.rate.sleep()  
            rospy.loginfo("Pose is %s %s", self.pose.x, self.pose.y)  
            rospy.loginfo("Speed is x: %s  theta: %s", vel\_msg.linear.x, vel\_msg.angular.z)

        # Wenn Ziel erreicht, Turtle stoppen  
        rospy.loginfo(" ######  Goal reached #######")  
        vel\_msg.linear.x = 0  
        vel\_msg.angular.z = 0  
        self.velocity\_publisher.publish(vel\_msg)

    def askForNewTry(self):  
        # Falls man noch zu anderen Koordinaten fahren möchte, erfolgt eine erneute Abfrage  
        tts = gTTS(text="Moechten Sie noch zu anderen Koordinaten fahren?", lang='de')  
        tts.save("audio12.mp3")  
        os.system("mpg321 audio12.mp3")  
        hearForNewTry = True  
        while hearForNewTry:  
            TryString = recognize\_speech\_from\_mic(recognizer, microphone)  
            TryString = str("{}".format(TryString["transcription"]))  
            if TryString == "ja":  
                hearForNewTry = False  
                turtle1.GoTurtle()  
            if TryString == "nein":  
                hearForNewTry = False  
                tts = gTTS(text="Mit Startseite, gelangen Sie zurueck zum Hauptmenue", lang='de')  
                tts.save("audio11.mp3")  
                os.system("mpg321 audio11.mp3")

    def GoTurtle(self):  
        turtle1.goal = turtle1.getGoalFromUser()  
        turtle1.move2goal()  
        turtle1.askForNewTry()

    def LinearTurtle(self):  
        turtle1.goal = turtle1.TurtleLinear()

    def \_RefreshVelocity(self):  
        # Aktualisieren von linea und angula velocity  
        global linea  
        global angula  
        global ActivateRefresh  
        vel\_msg = Twist()  
        ActivateRefresh = True  
        InstanceRefresh = True  
        while InstanceRefresh:  
            while ActivateRefresh:  
                vel\_msg.linear.x = linea  
                vel\_msg.angular.z = angula  
                self.velocity\_publisher.publish(vel\_msg)

    def SaveTrackData(self):  
        # Speichern der aktuellen Daten in Listen, zur Erfassung der Strecke  
        global linea  
        global angula  
        global SaveData  
        global LineaList  
        global AngulaList  
        global xList  
        global yList  
        global AnzahlLoops  
        startLoop = True  
        while startLoop:  
            while SaveData:  
                LineaList.append(linea)  
                AngulaList.append(angula)  
                xList.append(turtle1.pose.x\*50)  
                yList.append(turtle1.pose.y\*50)  
                AnzahlLoops += 1  
                print(LineaList)  
                time.sleep(0.05)

    def MenuAction(self):  
        # Startmenue - Fenster instanzieren  
        startseite = Startseite()  
        steuerung = Steuerung()  
        steuerung.close()  
        koordinaten = Koordinaten()  
        koordinaten.close()  
        richtung = Richtung()  
        richtung.close()  
        tts = gTTS(text="Hallo, dies ist eine Sprachsteuerung fuer die ros ToertelSim. Zur Auswahl einer Aktion reicht es diese Auszusprechen", lang='de')  
        tts.save("audio8.mp3")  
        os.system("mpg321 audio8.mp3")  
        hearformenu = True  
        while hearformenu:  
            MenuString = recognize\_speech\_from\_mic(recognizer, microphone)  
            MenuString = str("{}".format(MenuString["transcription"]))  
            print(MenuString)  
            if MenuString == "zu Koordinaten fahren":  
                koordinaten = Koordinaten()  
                startseite.close()  
                self.GoTurtle()  
            if MenuString == "in Richtung fahren":  
                richtung = Richtung()  
                startseite.close()  
                self.LinearTurtle()  
            if MenuString == "Steuerung":  
                steuerung = Steuerung()  
                startseite.close()  
            if MenuString == "beenden":  
                exit(0)  
            if MenuString == "startseite":  
                os.system("gnome-terminal -x rosrun turtlesim turtlesim\_node")  
                startseite = Startseite()  
                steuerung.close()  
                richtung.close()  
                koordinaten.close()

    def TurtleLinear(self):  
        global linea  
        global angula  
        global ActivateRefresh  
        global SaveData  
        global LineaList  
        global AngulaList  
        global AnzahlLoops  
        global startPainting  
        WartenAufAuswahl = False  
        startword\_is\_correct = False

        # Neuen Thread erstellen, der die Turtlesim position Aktualisiert  
        thread = threading.Thread(target=self.\_RefreshVelocity, args=())  
        thread.daemon = True  
        thread.start()

        # Startwort zum aktivieren des Programms  
        Startword = "starten"

        Trys = 1  
        MaxTrys = 5

        recognizer = sr.Recognizer()  
        microphone = sr.Microphone()

        tts = gTTS(text="Um das Programm zu aktivieren, bestaetige mit starten", lang='de')  
        tts.save("audio7.mp3")  
        os.system("mpg321 audio7.mp3")

        for i in range(Trys):  
            for j in range(MaxTrys):  
                calledWord = recognize\_speech\_from\_mic(recognizer, microphone)  
                strCalledWord = str("{}".format(calledWord["transcription"]))  
                if calledWord["transcription"]:  
                    break  
                if not calledWord["success"]:  
                    break  
                print("Ich konnte Sie nicht verstehen. Bitte erneute Eingabe\n")

            # wenn es einen Error gibt, neu instanzieren  
            if calledWord["error"]:  
                print("ERROR: {}".format(calledWord["error"]))  
                break

            # show the user the transcription  
            print("You said: {}".format(calledWord["transcription"]))

            if strCalledWord == Startword:  
                startword\_is\_correct = True

            if startword\_is\_correct:  
                print("Korrekt! Weiter im Programm".format(Startword))  
                tts = gTTS(text="Sie koennen nun die Schildkroete, durch die angegebenen Befehle steuern.", lang='de')  
                tts.save("audio2.mp3")  
                os.system("mpg321 audio2.mp3")  
                gotxqcord = True  
                anzahl = 0

                while gotxqcord:  
                    xqcord = recognize\_speech\_from\_mic(recognizer, microphone)  
                    zeichenx = str("{}".format(xqcord["transcription"]))

                    # Abfrage der Eingabe  
                    if zeichenx == "nach vorne" or zeichenx == "vorne":  
                        linea = 0.5  
                        angula = 0  
                        print("vorwaerts")  
                        SaveData = True  
                    if zeichenx == "nach hinten" or zeichenx == "hinten":  
                        linea = -0.5  
                        angula = 0  
                        print("rueckwaerts")  
                        SaveData = True  
                    if zeichenx == "nach rechts" or zeichenx == "rechts":  
                        linea = 0.5  
                        angula = -0.7  
                        print("rechts")  
                        SaveData = True  
                    if zeichenx == "nach links" or zeichenx == "links":  
                        linea = 0.5  
                        angula = 0.7  
                        print("links")  
                        SaveData = True  
                    if zeichenx == "Stillstand" or zeichenx == "Stopp" or zeichenx == "stehen bleiben" or zeichenx == "anhalten":  
                        linea = 0  
                        angula = 0  
                        print("Stillstand")  
                        SaveData = False

                    if zeichenx == "letzte Strecke":  
                        startPainting = True  
                        os.system("gnome-terminal -x rosrun turtlesim turtlesim\_node")  
                        MovesTrack = True  
                        tts = gTTS(text="Die letzte Strecke wird nun erneut gefahren.", lang='de')  
                        tts.save("audio13.mp3")  
                        os.system("mpg321 audio13.mp3")  
                        while MovesTrack:  
                            linea = LineaList[anzahl]  
                            angula = AngulaList[anzahl]  
                            anzahl += 1  
                            print("letzte Strecke wird ausgeführt")  
                            if anzahl >= AnzahlLoops:  
                                linea = 0  
                                angula = 0  
                                MovesTrack = False  
                                w.show()  
                                tts = gTTS(text="Sie koennen nun die Strecke speichern oder verwerfen", lang='de')  
                                tts.save("audio14.mp3")  
                                os.system("mpg321 audio14.mp3")  
                                WartenAufAuswahl = True  
                            time.sleep(0.05)

                    if zeichenx == "Strecke speichern" and WartenAufAuswahl == True:  
                        tts = gTTS(text="Legen Sie nun einen Datei namen fest", lang='de')  
                        tts.save("audio14.mp3")  
                        os.system("mpg321 audio14.mp3")  
                        print("Warte auf Dateinamen...")  
                        Dateiname = recognize\_speech\_from\_mic(recognizer, microphone)  
                        DateiString = str("{}".format(Dateiname["transcription"]))  
                        BildFormat = str(".png")  
                        print(DateiString+BildFormat)  
                        drawer.saveImage(DateiString+BildFormat, "PNG")  
                        tts = gTTS(text="Die Datei wurde erfolgreich unter"+DateiString+BildFormat+"gespeichert", lang='de')  
                        tts.save("audio15.mp3")  
                        os.system("mpg321 audio15.mp3")  
                        WartenAufAuswahl = False  
                        w.close()

                    if zeichenx == "Strecke verwerfen" and WartenAufAuswahl == True:  
                        WartenAufAuswahl = False  
                        w.close()  
                    if zeichenx == "schneller" and linea > 0:  
                        linea += 0.3  
                    if zeichenx == "schneller" and linea < 0:  
                        linea -= 0.3  
                    if zeichenx == "langsamer" and linea > 0:  
                        linea -= 0.3  
                    if zeichenx == "langsamer" and linea < 0:  
                        linea += 0.3  
                    if zeichenx == "Programm beenden":  
                        print("Programm beenden")  
                        os.system("mpg321 audio11.mp3")  
                        gotxqcord = False  
                        ActivateRefresh = False  
                        SaveData = False  
            else:  
                self.LinearTurtle()

class Steuerung(QMainWindow):  
    def \_\_init\_\_(self, parent=None):  
        super(Steuerung, self).\_\_init\_\_(parent)  
        # Steuerungs Fenster, mit angezeigter Informationsgrafik  
        self.setGeometry(600, 400, 700, 500)  
        self.setWindowTitle('Steuerung')  
        self.setWindowIcon(self.style().standardIcon(QStyle.SP\_FileDialogInfoView))

        vbox = QVBoxLayout()  
        label = QLabel(self)  
        label.setGeometry(0, 0, 700, 500)  
        pixmap = QPixmap('Steuerung.png')  
        label.setPixmap(pixmap)

        vbox.addWidget(label)  
        self.setLayout(vbox)  
        self.show()

class Koordinaten(QMainWindow):  
    def \_\_init\_\_(self):  
        super(Koordinaten, self).\_\_init\_\_()  
        # Zu Koordinaten fahren Fenster, mit angezeigter Informationsgrafik  
        self.setGeometry(600, 400, 700, 500)  
        self.setWindowTitle('Koordinaten')  
        self.setWindowIcon(self.style().standardIcon(QStyle.SP\_FileDialogInfoView))

        vbox = QVBoxLayout()  
        label = QLabel(self)  
        label.setGeometry(0, 0, 700, 500)  
        pixmap = QPixmap('Koordinaten.png')  
        label.setPixmap(pixmap)

        vbox.addWidget(label)  
        self.setLayout(vbox)  
        self.show()

class Richtung(QMainWindow):  
    def \_\_init\_\_(self):  
        super(Richtung, self).\_\_init\_\_()  
        # In Richtung fahren, mit angezeigter Informationsgrafik  
        self.setGeometry(600, 400, 700, 500)  
        self.setWindowTitle('Richtung')  
        self.setWindowIcon(self.style().standardIcon(QStyle.SP\_FileDialogInfoView))

        vbox = QVBoxLayout()  
        label = QLabel(self)  
        label.setGeometry(0, 0, 700, 500)  
        pixmap = QPixmap('Richtung.png')  
        label.setPixmap(pixmap)

        vbox.addWidget(label)  
        self.setLayout(vbox)  
        self.show()

class Drawer(QWidget):  
    def \_\_init\_\_(self, parent=None):  
        QWidget.\_\_init\_\_(self, parent)  
        # Widget instanzieren, abmaße festlegen  
        # PaintEvent zum Strecken speichern  
        self.setAttribute(Qt.WA\_StaticContents)  
        h = 550  
        w = 550  
        self.myPenWidth = 13  
        self.myPenColor = Qt.black  
        self.image = QImage(w, h, QImage.Format\_RGB32)

    def saveImage(self, fileName, fileFormat):  
        # Wird aufgerufen wenn Bild gespeichert werden soll  
        self.image.save(fileName, fileFormat)

    def paintEvent(self, event):  
        # Zeichen des Image in die BildBox  
        painter = QPainter(self)  
        painter.drawImage(event.rect(), self.image, self.rect())  
        self.image.fill(Qt.white)  
        self.paintEvent2(event)  
        painter.end()  
        self.update()

    def paintEvent2(self, event):  
        global linea  
        global angula  
        global positionx  
        global positiony  
        global AnzahlLoops  
        global xList  
        global yList  
        anzahl = 0  
        painter2 = QPainter(self.image)  
        painter2.setPen(QPen(Qt.black, 5, Qt.SolidLine))  
        painter2.setBrush(QBrush(Qt.blue, Qt.SolidPattern))

        # fuer jeden Punkt in der Liste, die die Positionen gespeichert haben, wird ein Rechteck an dessen Position gezeichnet  
        for i in range(AnzahlLoops):  
            anzahl += 1  
            positionx = xList[anzahl]  
            # Symmetrie umkehren, sonst ist das Bild im Vergleich zur Turtlesim spiegelverkehrt  
            positiony = 550 + (-1)\*yList[anzahl]  
            painter2.drawRect(positionx, positiony, 3, 3)

        self.update()

class BoxContainer(QWidget):  
    def \_\_init\_\_(self, parent=None):  
        QWidget.\_\_init\_\_(self, parent)

        # BildBox, hier wird das Image eingefügt, worauf der Painter die Strecke malt  
        self.setGeometry(0, 0, 570, 630)  
        self.setWindowTitle('Speicherung')  
        self.setStyleSheet("background-color: black;")  
        self.btnSave = QPushButton("Strecke speichern")  
        self.btnSave.setStyleSheet('background-color: rgb(0,255,68); color: #fff; font-size: 20px')  
        self.btnClear = QPushButton("Strecke verwerfen")  
        self.btnClear.setStyleSheet('background-color: rgb(252,23,3); color: #fff; font-size: 20px')  
        self.setLayout(QVBoxLayout())  
        self.layout().addWidget(self.btnSave)  
        self.layout().addWidget(self.btnClear)  
        self.layout().addWidget(drawer)

class Startseite(QMainWindow):  
    def \_\_init\_\_(self):  
        super(Startseite, self).\_\_init\_\_()  
        self.setGeometry(600, 400, 700, 500)  
        self.setStyleSheet("background-color: black;")  
        self.setWindowTitle('Startseite')

        layoutV = QVBoxLayout()  
        # Zu Koordinaten fahren Button  
        self.pushButton = QPushButton(self)  
        self.pushButton.setStyleSheet('background-color: rgb(0,238,255); color: #fff; font-size: 20px')  
        self.pushButton.setGeometry(200, 10, 300, 100)  
        self.pushButton.setText('Zu Koordinaten fahren')

        # In Richtung fahren Button  
        self.pushButton2 = QPushButton(self)  
        self.pushButton2.setStyleSheet('background-color: rgb(0,255,68); color: #fff; font-size: 20px')  
        self.pushButton2.setGeometry(200, 130, 300, 100)  
        self.pushButton2.setText('In Richtung fahren')

        # Steuerung Button  
        self.pushButton3 = QPushButton(self)  
        self.pushButton3.setStyleSheet('background-color: rgb(192,0,255); color: #fff; font-size: 20px')  
        self.pushButton3.setGeometry(200, 250, 300, 100)  
        self.pushButton3.setText('Steuerung')

        # Beenden Button  
        self.pushButton4 = QPushButton(self)  
        self.pushButton4.setStyleSheet('background-color: rgb(252,23,3); color: #fff; font-size: 20px')  
        self.pushButton4.setGeometry(200, 370, 300, 100)  
        self.pushButton4.setText('Beenden')

        # Zum Layout hinzufügen  
        layoutV.addWidget(self.pushButton)  
        layoutV.addWidget(self.pushButton2)  
        layoutV.addWidget(self.pushButton3)  
        layoutV.addWidget(self.pushButton4)  
        self.setLayout(layoutV)  
        self.show()

if \_\_name\_\_ == '\_\_main\_\_':  
    os.system("gnome-terminal -x roscore")  
    os.system("gnome-terminal -x rosrun turtlesim turtlesim\_node")  
    recognizer = sr.Recognizer()  
    microphone = sr.Microphone()  
    app = QApplication(sys.argv)  
    turtle1 = TurtleBotClass()  
    drawer = Drawer()  
    w = BoxContainer()  
    # Menue Thread wird gestartet, daemon sagt aus, dass das Hauptprogramm  
    # beendet werden darf, auch wenn noch Threads im Hintergrund laufen  
    threada = threading.Thread(target=turtle1.MenuAction, args=())  
    threada.daemon = True  
    threada.start()  
    # SaveTrackData Thread wird gestartet  
    threada2 = threading.Thread(target=turtle1.SaveTrackData, args=())  
    threada2.daemon = True  
    threada2.start()  
    sys.exit(app.exec\_())