6. Programmierung

Der Schreitroboter hat eine offene API, d.h. die Möglichkeiten, den Roboter mit eigener Programmierung auszustatten und in eigenen Projekten zu integrieren sind groß. Hierbei gibt es zwei grundlegende Möglichkeiten, die allerdings auch verbunden werden können.

Die eine Möglichkeit ist die direkte Programmierung mit der Spot-API. Dies ist insbesondere dann notwendig, wenn neue Services entwickelt werden, da hierzu gRPC und die ProtoBuf-Nachrichten genutzt werden sollten, um die Schnittstellen mit möglichst vielen Programmiersprachen kompatibel zu halten.

Die zweite Möglichkeit ist die Programmierung mittels der, von Boston Dynamics mitgelieferten, Python-Library. Diese Bibliothek vereinfacht die Nutzung, da der Anwender direkt in Python programmieren und mit gRPC und ProtoBufs im Normalfall selten in Berührung kommt.

In größeren Projekten werden diese beiden Möglichkeiten vermutlich zusammen genutzt, insbesondere wenn zusätzlich zur Standardnutzung des Roboters neue Services definiert werden müssen. Dies kann notwendig sein, um neue Payloads, wie z.B. einen GPS-Empfänger, zu integrieren, da der Roboter werkseitig hierfür keine Unterstützung bietet.

6.1 Programmierung mit der Python Library unter Windows

Die Python-Library der Spot-API von Boston Dynamics kann genutzt werden, um mit dem Spot-Python-SDK (Software Development Kit) Anwendungen zu entwickeln mit denen der Roboter zu kontrollieren. Um mit der Spot-Python-SDK arbeiten zu können wird Python 3.6 oder Python 3.7 benötigt. Python 3.8 und neuer wird nicht unterstützt.

Zusätzlich wird der Paket-Installierer „pip“ benötigt. Dieser ist im Normalfall in jeder Version von Python 3.4 und größer enthalten. Wenn dies nicht der Fall ist, muss „pip“ manuell hinzu installiert werden.

Anschließend können mit „pip“ mit dem folgenden Befehl die benötigten Python-Pakete von Boston Dynamics heruntergeladen werden:

python -m pip install –-upgrade bosdyn-client bosdyn-mission bosdyn-choreography-client

Nun kann mit folgendem Befehl verifiziert werden, dass die benötigten Pakete installiert wurden:

python -m pip list –format=columns | findstr bosdyn

Nun sollte eine Ausgabe wie auf folgendem Bild angezeigt werden:

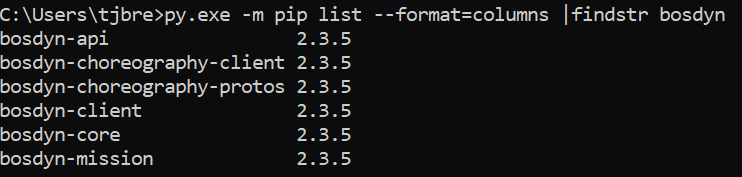


Abbildung - Anzeige installierter Boston Dynamics Pakete

Die derzeitige Version der Spot-API ist die 2.3.5. Wenn eine neue Version verfügbar ist, muss diese zur Nutzung sowohl auf dem zur Programmierung genutzten Computer als auch auf allen genutzten Robotern und Payloads aktualisiert werden.

Wenn mit dem Roboter nur bereits programmierte Lösungen genutzt werden sollen, ist die Installation dieser Pakete ausreichend. Wenn aber Lösungen selbst programmiert werden sollen, ist ein Download des kompletten Spot-SDK nötig. Dieses enthält zusätzlich Programmierbeispiele, ProtoBuf-Definitionen und die Dokumentation der API. Das Spot-SDK ist auf Github unter folgendem Link erhältlich:

<https://github.com/boston-dynamics/spot-sdk>

Anwender können das SDK entweder direkt von Github herunterladen oder dieses innerhalb einer IDE ( Integrated Development Environment) tun. Im Rahmen dieser Praxisarbeit wurde die IDE PyCharm in der Community Edition, Version 2021.1.2, der Firma JetBrains verwendet. In dieser wurde unter dem Punkt Git die SDK heruntergeladen. Zusätzlich muss in den Einstellungen der richtige Python-Interpreter ausgewählt werden. Dieser muss dem Python-Interpreter entsprechen, mit dem die Spot-API heruntergeladen wurde. Zusätzlich müssen, je nach ausgewähltem Beispiel weitere Pakete nachinstalliert werden.

6.2 Beispielprogramme mit der Python Library in der Online-API und dem downloadbaren Spot-SDK (Software Development Kit)

Das Python-SDK enthält viele Beispielprogramme, die einen großen Teil des Funktionsumfang des Schreitroboters umfassen. Dies geht von grundlegenden Beispielen, die die Kommunikation mit dem Roboter herstellen und grundlegende Bewegungen realisiert (hello\_spot.py), über die Steuerung des Roboters per Tastatur oder Xbox-Kontroller (wasd.py, xbox-controller.py), beispielhaften Durchführungen von autonomen Missionen, die im Voraus aufgezeichnet und dann wieder abgespielt werden (mission\_recorder.py, replay\_mission.py) bis hin zu einem Beispiel in dem der Roboter durch maschinelles Lernen eine Person oder einen Gegenstand erkennen und folgen kann. In der Online-API ist zudem ein Beispiel enthalten, dass zeigt, wie man den Roboter programmieren muss, damit man diesem „Stöckchen holen“ spielen kann.

Für den Anwender zu Begin sehr wichtige Beispiele, da Sie die grundlegenden Funktionen des Roboters demonstrieren sind:

* hello-spot.py: Einfaches Beispiel, um sich mit einem Spot zu verbinden und Befehle an diesen zu schicken
* estop\_nogui.py, estop\_gui.py: Beispiel zum Herstellen eines E-Stops
* get\_robot\_state.py: Beispiel zum Abfragen der Statusdaten des Roboters
* get\_image.py: Beispiel zum Abrufen und Anzeigen von Kamerabildern
* wasd.py: Beispiel zur Steuerung des Roboters durch Senden von Kommandos, die über die Tastatur ausgelöst werden, sowie eine Anzeige der verfügbaren Befehle und Kamerabildern als ASCII-Repräsentation auf der Kommandozeile

Bei vielen Beispielen werden zusätzliche Python-Pakete benötigt. Hierfür ist im Ordners eines jeden Beispielprogramms eine Datei mit dem Namen „requirements.txt“ vorhanden. Um die Pakete zu installieren, muss man in der Kommandozeile in diesen Ordner navigieren und folgenden Befehl ausführen:

py.exe -m pip install -r requirements.txt

Anschließend werden die benötigten Pakete installiert und das Beispiel kann ausgeführt werden.

6.3 Notwendige Befehlsaufrufe

Zur Programmierung des Roboters gibt es einige Funktionsaufrufe, die notwendig sind, bevor Daten abgerufen oder Befehle an den Roboter gesendet werden können.

6.3.1 Erstellen eines SDK-Objekts

Alle Anwendungen müssen damit starten, dass ein SDK-Objekt generiert wird, welchem ein Name gegeben wird. Dieser hat keine semantische Bedeutung, sondern dient nur zum Debuggen. Somit kann jeder String genutzt werden. Das SDK-Objekt wird durch folgenden Aufruf erstellt:



Abbildung Erstellen der SDK

In diesem Fall wurde der Name „Praxisarbeit“ gewählt.

6.3.2 Erstellen eines Roboter-Objekts

Als nächstes wird ein Roboter-Objekt erstellt. Aus diesem werden im späteren Verlauf alle weiteren Clients generiert, mit denen der Roboter angesprochen wird. Das Roboter Objekt wird mit folgenden Funktionsaufruf aus dem SDK-Objekt generiert:



Abbildung Erstellen des Roboter-Objekts

Als „hostname“ ist hier die IP-Adresse des Roboters zu verwenden. Die ermöglicht es, in einer Anwendung mehrere Roboter zu erstellen und zu kontrollieren.

6.3.3 Authentifizierung des Nutzers am Roboter

Um die meisten Befehle nutzen zu können, wie z.B. der Roboter zu steuern oder auch Statusdaten vom Roboter anzufragen, muss sich die Anwendung beim Roboter authentifizieren. Hierfür wird folgender Funktionsaufruf genutzt:



Abbildung Authentifizierung der Anwendung am Roboter

Zur Identifizierung werden eine Benutzername und ein Passwort benötigt. Die zum Auslieferungszeitpunkt vergebenen Daten sind im Batteriefach des Roboters zu finden.

Es gibt einige wenige Funktionen des Roboters die auch ohne die Authentifizierung ausgeführt werden können. Zu diesen gehören:

* Abfragen der „robot-id“
* Anzeigen der auf dem Roboter verfügbaren Services

6.3.4 E-Stop

Der Emergeny-Stop (Notaus, kurz estop) muss genutzt werden, damit bei einer Fehlfunktion des Roboters oder der kontrollierenden Anwendung Sicherheit gegenüber allen beteiligten Personen gewährleistet werden kann. Zuerst wird mit folgendem Aufruf eine E-Stop-Client erstellt:



Abbildung Erstellen des E-Stop-Client

In diesem Fall wird hier die Bezeichnung „EstopClient.default\_service\_name“ übergeben. Hinter dieser verbirgt sich der String „estop“. Einen solchen „default\_service\_name“ gibt es für jeden Client, was die Erstellung der Service-Clients vereinfacht.

Als nächstes muss eine E-Stop-Endpoint generiert und am Roboter angemeldet werden. Es muss mindestens ein E-Stop-Endpoint am Roboter angemeldet werden, sonst können die Motoren nicht angeschaltet werden.

SICHERHEITSHINWEIS: Während ein E-Stop-Endpoint am Roboter angemeldet wird, wird ein Notaus am Roboter ausgelöst. Diese Aktion sollte nur ausgeführt werden, wenn die Motoren des Roboters ausgeschaltet sind.

Der E-Stop-Endpoint wird mit folgenden Funktionsaufrufen erstellt und angemeldet:



Abbildung Erstellen des E-Stop-Endpoint



Abbildung Anmelden des E-Stop-Endpoint am Roboter

Der E-Stop-Endpoint muss regelmäßig ein Signal an den Roboter senden, damit die Motoren angeschaltet werden dürfen. Hierzu wird die EstopKeppAlive-Klasse verwendet mit folgendem Funktionsaufruf:



Abbildung Erstellen des EstopKeepAlive- Objekts



Abbildung Starten des EstopKeepAlives

6.3.5 Lease

Bevor die Motoren angeschaltet werden können, muss die Anwendung Besitz vom Roboter ergreifen. Dies liegt daran, dass mehrere Anwendungen auf den Roboter zugreifen können, aber nur die Anwendung, die den Roboter „besitzt“, kann ihn kontrollieren. Hierfür muss ein Lease-Client erstellt werden. Dies geschieht mit dem folgenden Funktionsaufruf:



Abbildung Erstellen des Lease-Clients

In diesem Fall wird hier die Bezeichnung „LeaseClient.default\_service\_name“ übergeben. Hinter dieser verbirgt sich der String „lease“. Einen solchen „default\_service\_name“ gibt es für jeden Client, was die Erstellung der Service-Clients vereinfacht.

Als nächstes muss die Lease für den Roboter erworben werden, sowie, wie beim E-Stop-Endpoint, ein KeepAlive-Objekt definiert werden, damit die Anwendung regelmäßig die Lease „erneuert“. Wenn dies nicht passiert, verhält sich der Roboter wie beim Verlust des E-Stop-KeepAlive, wird also die Motoren abschalten. Hierzu werden folgende Funktionsaufrufe benötigt:



Abbildung Erwerben der Lease und erstellen des Lease-KeepAlive-Objekts

6.3.6 Time-Sync

Um den Unterschied in den Uhrzeiten zwischen der Anwendung und dem Roboter auszugleichen, muss ein timesync stattfinden. Hierzu bietet die robot-Klasse einen timesync-Thread, der über folgenden Funktionsaufruf aufgerufen wird:



Abbildung Timesync etablieren

Dieser Aufruf ist ein blockierender Funktionsaufruf, der die weitere Abarbeitung der Anwendung blockiert, bis der Timesync hergestellt ist.

6.4 Selbstprogrammierte Anwendung

Die Aufgabe beinhaltet die Programmierung einer Anwendung für folgende Aufgaben. Die Abfrage von Daten, wie z.B. Statusdaten und grundlegende Informationen über den Roboter, sowie das Senden von grundlegenden Befehlen an den Roboter, wie z.B. die Befehle zum Starten und bewegen des Roboters. Zusätzlich ist in der programmierten Software die Funktionalität enthalten, Kamerabilder vom Roboter zu Empfangen. Diese werden periodisch mit einem Bild pro Sekunde vom Roboter angefordert und angezeigt. Zudem kann ausgewählt werden, von welcher Kamera das Bild angezeigt werden soll.

6.4.1 Benutzeroberfläche

Die Funktionalität der Anwendung wird dem Anwender durch eine Benutzeroberfläche bereitgestellt. Für diese Benutzeroberfläche wurde PySide2 verwendet eine Python-Version des Qt5 Toolkits, eines Open Source Widget Toolkits, zur Erstellung von grafischen Benutzeroberflächen.

Die Benutzeroberfläche des in der Praxisarbeit entwickelten Programms gliedert sich in zwei Teile:

Der erste Teil ist das Login-Fenster. Dieses bietet drei Eingabefelder. In diese werden der Benutzername, das Passwort und der Hostname des Roboters eingegeben. Anschließend wird auf den Login-Button geklickt. Wenn die Daten korrekt sind, und eine Verbindung zum Roboter hergestellt werden konnte öffnet sich das nächste Fenster. Wenn eins oder mehrere der Eingabefelder bei einem Klick auf den Login-Button leer sind, wird in einer Info-Box der Text „At least one credential is missing“ angezeigt. Wenn die eingegebenen Daten falsch sind, oder keine Verbindung zum Roboter hergestellt werden konnte wird in der Info-Box der Text „Something went wrong, either communication with the robot or you entered wrong credentials“ angezeigt.

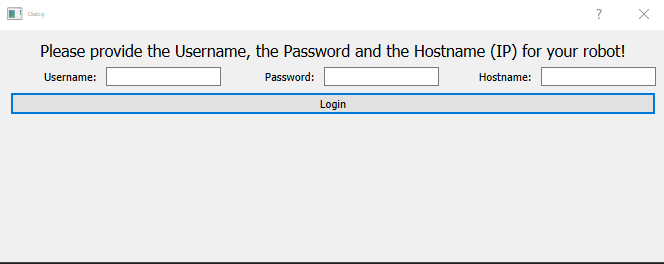


Abbildung 13 Login-Fenster

Das Login-Fenster ist in der Python-Datei gui\_loin\_screen.py definiert.

Der zweite Teil ist das Haupt-Fenster, in dem der Roboter gesteuert wird. Dieses ist in drei Teilen aufgebaut. Im linken Teil befindet sich der „Requesting-Data“-Teil. Hier kann ausgewählt werden welche Daten vom Roboter abgerufen werden sollen. Zur Auswahl stehen:

* ID
* Status
* Hardware-Info
* Metrics

Mit einem Klick auf den „Send Request“-Button werden die Daten abgerufen und anschließend in der TextBox angezeigt.

Im mittleren Teil wird ein Kamerabild des Roboters angezeigt. Dieses Kamerabild wird ein Mal pro Sekunde vom Roboter angefordert und anschließend. Es kann aus den Kamerabildern der am Roboter fest verbauten Fisheye-Kameras gewählt werden, welches Bild angezeigt wird.

Im rechten teil des Fensters ist die Steuerung des Roboters verortet. Hier kann der Roboter gestartet und gestoppt, die Motoren an- und abgeschaltet und der E-Stop an- und ausgeschaltet werden. Zudem sind für diese drei Funktionalitäten drei Felder eingefügt ,die den derzeitigen Status anzeigen. Diese sind rot und enthalten das Wort „OFF“ wenn die Funktionalität ausgeschaltet ist und sind grün und enthalten das Wort „ON“ wenn die Funktionalität angeschaltet ist. Diese Felder werden, zusammen mit dem Bild des mittleren Teil des Fensters, ein Mal pro Sekunde aktualisiert. Weiterhin kann der Roboter in alle Richtungen bewegt werden, sowie Sitzen und Stehen gelassen werden.

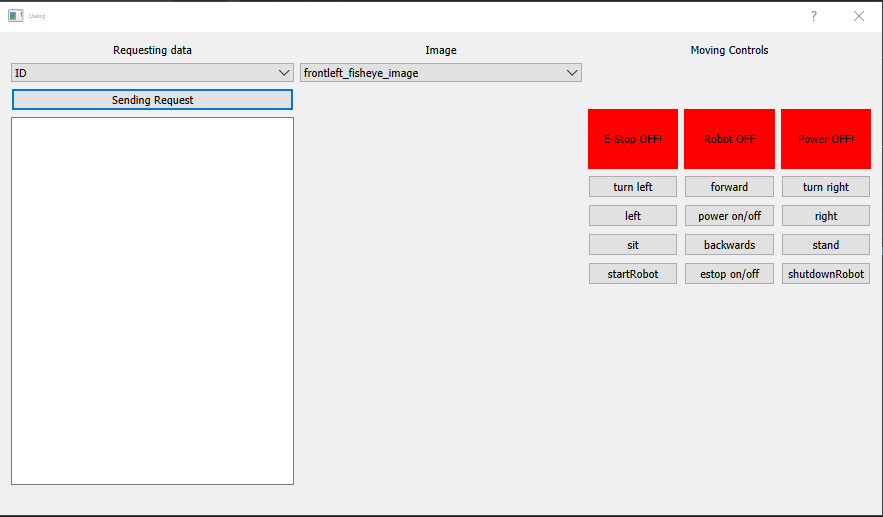


Abbildung 14 Haupt-Fenster

Das Haupt-Fenster ist in der Datei gui\_main\_screen.py definiert.

Falls das Haupt-Fenster geschlossen wird, während die Motoren des Roboters angeschaltet sind und der Roboter eventuell gerade eine Aktion ausführt, bleibt der Roboter stehen, setzt sich hin und schaltet die Motoren ab. Dies wird durch ein Überschreiben der „closeEvent()“-Funktion des Haupt-Fensters erreicht, wobei in der neu definierten Funktion das Power-Off Kommando an den Roboter gesandt wird, mit dem Zusatz „cut\_immediately=false“, damit die Motoren nicht sofort abgeschaltet werden:

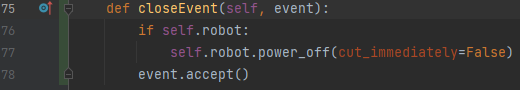


Abbildung 15 Methodenaufruf bei Schließen des Fensters

Beide Fenster werden als Klassen in der Datei GUI.py zusammengeführt und initialisiert, wobei alle Buttons und Eingabe- sowie Ausgabe-Fenster und Boxen die mit den entsprechenden Funktionen zum Auslesen der Fenster, Verarbeitung der Daten und Ausgabe der Ergebnisse verknüpft werden.

Zudem wird in der Klasse MainScreen für das Haupt-Fenster ein Timer gestartet, der einmal pro Sekunde eine Funktion aufruft, die das angeforderte Bild vom Roboter lädt, sowie die Stati der Motoren, des E-Stops und der Lease abfragt.

6.4.2 Abfrage der Daten

Die Abfrage der Daten findet über zwei Services statt. Den Robot-ID-Service und den Robot-State-Service welcher die Abfrage von Status, Hardware-Info und Metriken ermöglicht.

Die Methoden hierzu befinden in der Python-Datei „data-aquisition.py“. Als Beispiele sind in den folgenden Bildern die Abfrage der Robot-ID und des Robot-State gezeigt:

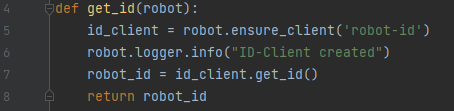


Abbildung 16 Abfrage der Robot-ID

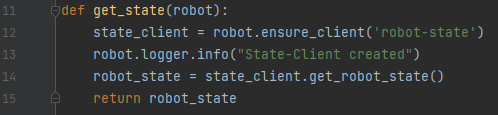


Abbildung 17 Abfrage des Robot-State

In beiden Fällen wir zuerst ein entsprechender Client erstellt. Auf diesem wird die Get-Methode aufgerufen, durch welche die Daten vom Roboter angefordert werden. Anschließend werden die Daten an die aufrufende Methode der Benutzeroberfläche zurückgegeben. Für die Abfrage der Hardware-Info sowie der Metriken wird auch ein Robot-State-Client erstellt und die Zeile 14 abgeändert in „.get\_robot\_hardware\_configuration()“ und „.get\_robot\_metrics()“.

Die Daten die Roboter zurückschickt sind als ProtoBuf-Nachrichten kodiert. Aus diesen könnten die gewünschten Teildaten extrahiert werden. Z.B. könnte aus dem zurückgegebenen Robot-State der Batterieladungszustand ausgelesen und z.B. als ein Statusbar auf einer Oberfläche angezeigt werden. Der Einfachheit halber wurden die ProtoBuf-Nachrichten in der vorliegenden Anwendung nur in einer TextBox ausgegeben. Hierzu wird die ProtoBuf-Nachricht in eine String gewandelt. Dies geschieht mit der Python-Methode „str()“. Anschließend kann dieser String mit der „setText“-Methode der TextBox ausgegeben werden:



Abbildung 18 Wandlung und Anzeige des Datenstrings

6.4.3 Anzeige des Kamerabildes

Im mittleren Teil des Haupt-Fenster der Anwendung wird ein Bild einer der festverbauten Kameras des Roboters angezeigt. Dieses wird ein Mal pro Sekunde aktualisiert. Der Anwender kann mittels eines Dropdown-Menus auswählen von welcher der angezeigten Fisheye-Kameras angezeigt werden soll.

Die Funktionalität das Bild vom Roboter zu laden ist in der Python-Datei „camera\_image.py“ in der Methode „get\_image()“ vorhanden. Hier wird, wie im vorangegangenen Teil ein Service-Client erstellt. Dieses Mal ein Image-Service-Client:



Abbildung 19 Erstellen des Image-Service-Clients

„image\_service“ hat in dieser Methode de Default-Wert „ImageCLient.default\_service\_name“ welcher den String „image\_service“ enthält.

Als nächstes wird auf diesem Image-Service-Client die Funktion „get\_image\_from\_sources()“ aufgerufen, welcher die vorher ausgewählte Bildquelle übergeben wird:



Abbildung 20 Abfragen des Bildes

Anschließend wird das Bild mit der Open-Source Bildbearbeitungsbibliothek bearbeitet und gespeichert. Zur Anzeige gebracht wird das Bild über das Widget „QPixmap“ von PySide2, welches das Bild lädt und auch einem Label anzeigt:

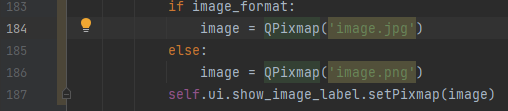


Abbildung 21 Laden und Anzeigen des Bildes

6.4.4 Steuerung des Roboters

Zur Steuerung des Roboters dient das rechte Drittel des Haupt-Fensters. Hierzu wird, nach dem Login-Vorgang ein Objekt der Klasse „MoveRobot“ erstellt. Diese Klasse ist in der Python-Datei „move\_robot.py“ definiert und enthält alle Methoden zur Steuerung des Roboters.

Bei der Initialisierung dieses Objekts werden alle benötigten Service-Clients erstellt. Diese sind:

* Lease-Client
* E-Stop-Client
* Power-Client
* Kommando-Client

Die folgenden Bilder zeigen die Methodenaufrufe zu Erstellung des jeweiligen Services:



Abbildung Erstellen des Lease-Client



Abbildung Erstellen des E-Stop-Clients und des E-Stop-Endpoints



Abbildung Erstellen des Power-Clients



Abbildung Erstellen des Kommando-Clients

Bevor der Roboter gesteuert werden kann, muss die Anwendung die „Gewalt“ über den Roboter an sich nehmen. Dies geschieht durch einen Klick auf den Button „startRobot“. Dadurch wird die start-Methode des „MoveRobot“-Objekts aufgerufen. In diesem wird eine Lease auf den Roboter aufgenommen und ein LeaseKeepAlive-Thread gestartet, der diese Lease periodisch erneuert. Zusätzlich wird der vorher generierte E-Stop-Endpoint initiaisiert. D

6.5 Ausblick auf weitere Möglichkeiten der Programmierung