

# Rescue Robot

## Projekt angewandte Elektrotechnik

**Bruno Berger**

**Matrikelnr.2170706**

Hochschule Hamm-Lippstadt  
Interaktionstechnik und Design  
Dr.-Arnold-Hueck-Straße 3  
59557 Lippstadt

Email: bruno.berger@stud.hshl.de

**Lukas Walter**

**Matrikelnr.2170086**

Hochschule Hamm-Lippstadt  
Interaktionstechnik und Design  
Dr.-Arnold-Hueck-Straße 3  
59557 Lippstadt

Email: lukas.walter@stud.hshl.de

**Melanie Löbel**

**Matrikelnr.2170582**

Hochschule Hamm-Lippstadt  
Interaktionstechnik und Design  
Dr.-Arnold-Hueck-Straße 3  
59557 Lippstadt

Email: melanie.loebel@stud.hshl.de

**Abstract**—Dieses Dokument beschreibt die Entwicklung eines Rettungsroboters, welcher in Fällen wie Naturkatastrophen oder schweren Industrieunfällen zum Einsatz kommt. Hier ist es meist dringend notwendig, verletzte Personen zu retten oder Gegenstände zu bergen. Um dabei nicht andere Menschen den Risiken oder Gefahren auszusetzen, ist es sehr hilfreich, Roboter einzusetzen. In diesem Projekt soll ein "Rescue-Robot" entwickelt werden, der bei einem Industrieunfall wie einer Explosion seinen Einsatz findet. Bei der Explosion wurden auch radioaktive Objekte auf dem Industriegelände verteilt. Der Rescue-Robot soll sich mittels gesendeter Radiosignale autonom auf dem Industriegelände bewegen und beim Erkennen radioaktiver Objekte, diese bergen. Beim Erkennen einer Person soll über den Rescue-Robot eine Kommunikation zwischen dieser und dem Rettungsdienst von außen möglich sein. Für die model-basierte Entwicklung wurden Werkzeuge wie Solidworks (3D-Entwurf), Enterprise Architecture (Erstellung von Diagrammen), C# und Unity (Laufzeit- und Entwicklungsumgebung) für die Visualisierung verwendet.

## 1. Einführung

*Löbel*

Die Domäne Rettungsrobotik verfügt über sehr vielversprechende wirtschaftliche Aspekte. Menschliche Rettungskräfte sind bei Katastrophen Szenarien eine knappe Ressource. Ein einzelner Bediener sollte daher idealerweise eine Vielzahl von Robotern überwachen. Beim Einsatz solcher Rettungsroboter ist ein hohes Maß an Autonomie gefordert. Sie begegnen einer Vielzahl unbekannter Objekte. Sie sollten in der Lage sein, Objekte oder ihre Umgebung zu erkennen und spezifiziert darauf zu reagieren. Es gibt zwei Entwicklungsziele, die schwer zu kombinieren sind. Zum einen ist für die Umsetzung eines autonomen Systems High-Technology notwendig. Zum anderen gibt es den Bedarf möglichst einfache und kostengünstige Systeme zu entwickeln [1]. Bei diesem Projekt des "Rescue Robots" liegt der Hauptfokus zunächst auf der Umsetzung der Funktionalität.

August 27, 2020

## 1.1. Produktübersicht

Der Rescue Robot soll auf einem Industriegelände, auf dem es eine Explosion gab, eingesetzt werden. Auf dem Gelände können sich noch Personen befinden, die eventuell verletzt sind. Dies bedeutet, dass auch ein Rettungsdienst vor Ort sein muss. Dieser soll über den Rescue Robot mit verletzten Personen kommunizieren können. In Abb.1 wird dargestellt, in welchem Kontext der Rescue Robot seinen Einsatz findet.

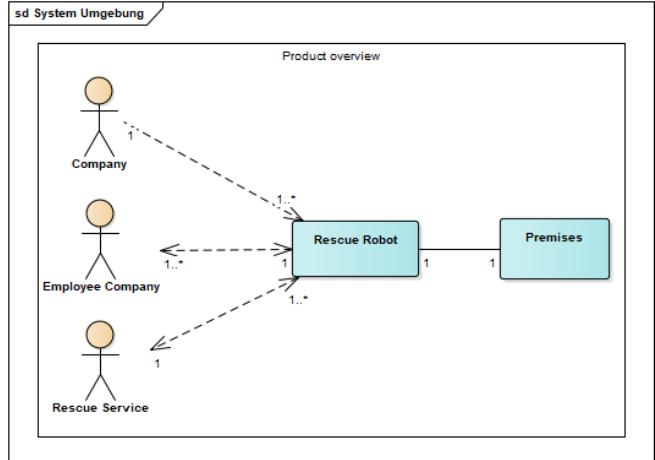


Figure 1. Produktübersicht

## 1.2. Systemumgebung

Der Rescue Robot soll das Firmengelände autonom erkunden können. Hierfür soll er Radiosignalen folgen können. Diese werden von Funktürmen auf dem Firmengelände gesendet. Das Firmengelände ist durch Mauern begrenzt. Durch die Explosion befinden sich auf dem Gelände unter anderem Gesteinsbrocken, welche umfahren werden müssen. Aber auch Gegenstände, die je nach radioaktiver Strahlung, aufgesammelt werden müssen. Das

Firmengelände besteht hauptsächlich aus festem Untergrund. Jedoch ist durch die Explosion auch ein Wasserloch entstanden, welches der Roboter überqueren können soll. Zusätzlich wird die Sicht durch Nebel oder Rauch erschwert. Außerdem können sich verletzte Personen auf dem Gelände befinden. In Abb.2 ist die Systemumgebung, das Firmengelände "Premises" als Klassendiagramm dargestellt.

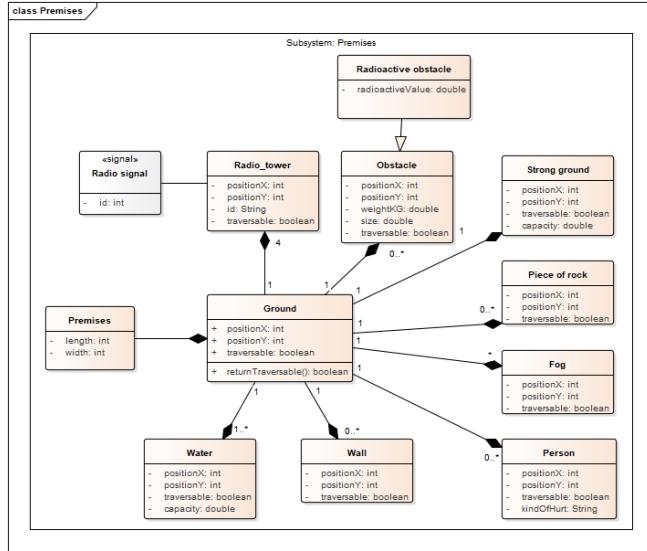


Figure 2. Systemumgebung

### 1.3. Anforderungen

Aus dem beschriebenen Szenario lassen sich folgende Anforderungen festhalten, siehe Abb.3.

Category	ID	Requirement
Fortbewegung	FR1	Fahrzeug muss sich an Land fortbewegen können
	NFR1	Fortbewegung muss über Ketten laufen
	FR2	Fahrzeug muss sich auf der Wasseroberfläche fortbewegen können
	NFR2	Fortbewegung muss über eine Turbine und ein Ruder laufen
Gegenstände bergen	FR3	Das Fahrzeug muss Gegenstände greifen können
	NFR3.1	Das Fahrzeug muss über einen beweglichen Greifer verfügen
	FR4	Das Fahrzeug muss über einen verschließbaren Behälter verfügen
	NFR5	Das Fahrzeug muss über einen offenen Behälter verfügen
Sensorik	FR6	Das Fahrzeug muss Radiosignale folgen können
	FR7	Das Fahrzeug muss erkennen ob es im Wasser ist
	NFR8	Das Fahrzeug muss über geeignete Peripherie verfügen um mit Menschen zu kommunizieren
Wartbarkeit	NFR9	Motoren oder Turbine müssen leicht zugänglich sein.

Figure 3. Requirements

Die funktionalen Anforderungen ergeben folgendes Use Case Diagram (Abb.4), welche das System beschreiben. Der Rescue Robot soll sich sowohl an Land als auch im Wasser fortbewegen können. Hierzu muss er mittels eines Sensors erkennen können, ob er sich an Land oder im Wasser befindet. Zudem muss der Rescue Robot Radiosignale empfangen und diesen folgen können. Trifft der

Roboter auf radioaktive Objekte sollen diese eingesammelt werden. Dazu muss der Roboter über einen beweglichen Greifer verfügen und greifen können. Außerdem soll mit Personen, die sich noch auf dem Gelände befinden, kommuniziert werden können. Der Roboter muss deshalb über die geeignete Peripherie wie Kamera, Lautsprecher und Mikrofon verfügen.

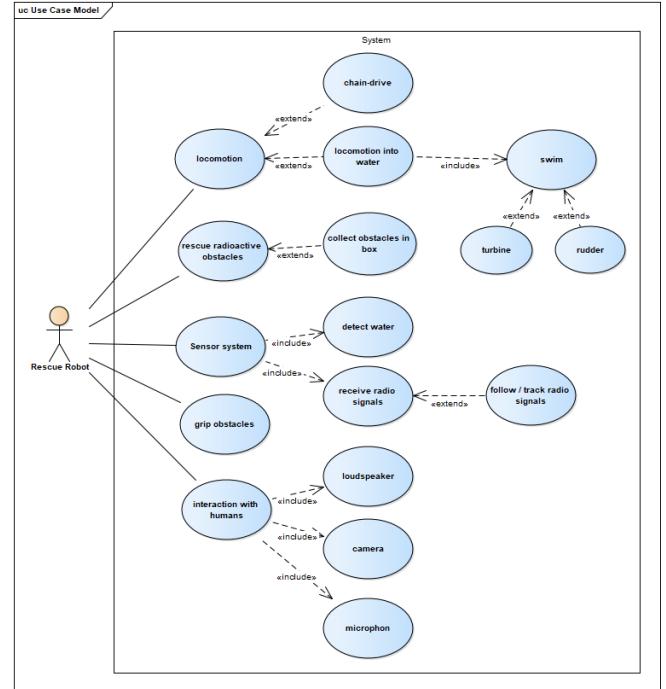


Figure 4. Use Case Diagram

## 2. Entwurf

Löbel

Nach Festlegung der funktionalen Anforderungen wurde ein erster Prototyp im 3D erstellt. Grundlage hierfür waren einzelne Paper Prototypes der Gruppenmitglieder. Nach Bewertung der einzelnen Prototypes wurden Teile bzw. Komponenten übernommen oder ergänzt.

### 2.1. 3D Prototype

Der resultierende Prototyp, siehe Abb.5, beinhaltete wie in den Anforderungen bereits erwähnt, einen Kettenantrieb für die Fortbewegung an Land (NFR1). Für die Fortbewegung im Wasser wurden Turbine und Ruder (NFR2) zentral im unteren Bereich des Roboters in einem Durchgangsloch platziert. Für das Bergen von Gegenständen wurde ein schwenkbarer Greifarm mit Greifer zentral an der Fahrzeug Vorderseite positioniert (NFR3.1). Der Greifer wurde von der GrabCAD Library importiert, <https://grabcad.com/library/4-bar-linkage-gripper-with-dynamixel-rx-64-1>. Zur zusätzlichen Unterstützung beim Greifprozess wurden im Prototypen zwei herausfahrbare Stützen an der Fahrzeug-Vorderseite angebracht. Zum Sammeln der radioaktiven

Gegenstände befindet sich im hinteren Teil des Roboters eine verschließbare Box (FR4). Daneben wurde eine offene Box (NFR5) eingebracht zum Transport von beispielsweise "Erste-Hilfe" Material oder einem Koffer. Vorne links am Fahrzeug wurden LIDAR Sensor inklusive Peripherie wie Kamera, Lautsprecher und Mikrofon auf einem Drehpodest platziert (NFR8). Zusätzlich ist dieser Teil schwenkbar.

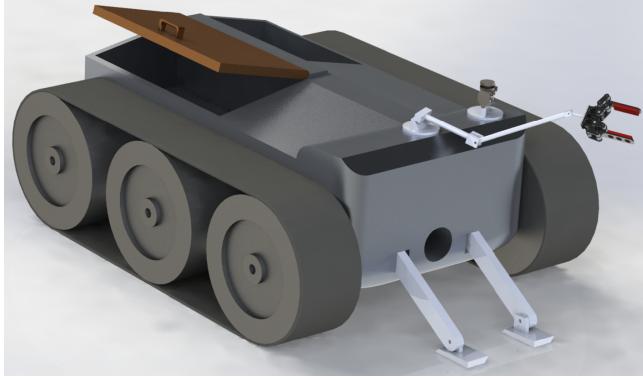


Figure 5. Kombinierter 3D Prototyp

### 3. Konzept Löbel

Anschließend wurden über die objektorientierte Analyse die benötigten Klassen und Attribute definiert.

#### 3.1. Technische Systemarchitektur

Das technische System besteht aus folgenden Komponenten: Die Sensorik, die für das Senden und Empfangen von Signalen verantwortlich ist. Hier zählen unter anderem die Antennen zum Empfangen von Radiosignalen dazu. Ein LIDAR Sensor, über den Laserimpulse gesendet und reflektiertes gestreutes Licht bei Objekterkennung empfangen wird. Ein Wassersensor zur Messung der Kapazität, ob der Roboter sich an Land oder Wasser befindet (FR7) und ein Kraft- und Druckaufnehmer zur Gewichtsmessung der zu bergenden Gegenständen gehören auch dazu. Die Signalberechnung bzw. -wertung wird durch einen Microcontroller übernommen. Je nach empfangenen Signalwert werden die jeweiligen Motoren des Roboters gesteuert. Es werden unter anderem Motoren für den Greifer und Greifarm benötigt, als auch für den Antrieb von Ketten, Turbine, Ruder, das Drehen und Schwenken der Peripherie oder auch für die ausfahrbaren Stützen und den verschließbaren Deckel der Bergungsbox. Hinzu kommt die Peripheriesteuerung für die Kommunikation, die sich aus den Objekten Kamera, Lautsprecher und Mikrofon zusammensetzt.

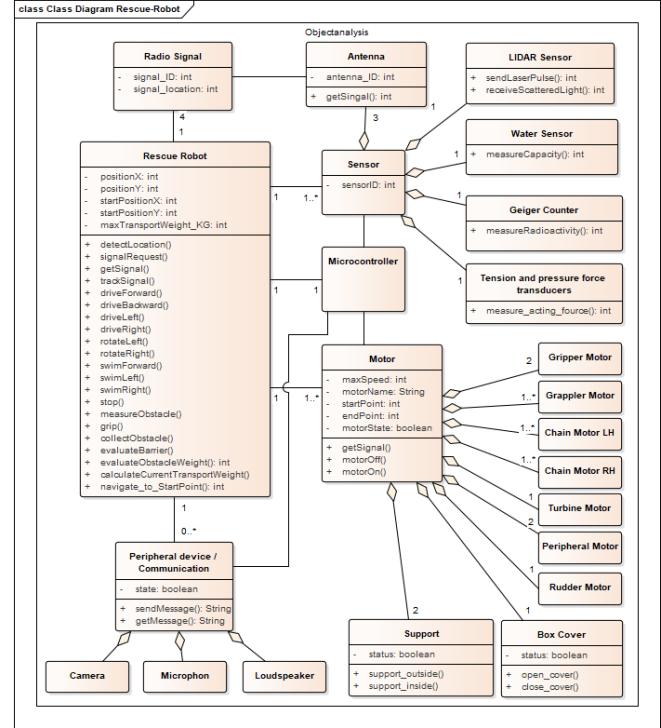


Figure 6. Class Diagram "Rescue Robot"

#### 3.2. Subsysteme

Nach Aufteilung der Objekte in die Kategorien "Signal transmit", "Signal reception", "Signal calculation", "Motor control" und "Peripheral device control" wurde eine Swimlane Analyse durchgeführt, in der der gesamte Prozessablauf beschrieben wurde. Daraus ergaben sich die folgenden Subsysteme:

- das Firmengelände (siehe Abb. 2)
- die Signalverfolgung
- die Fortbewegung (an Land und im Wasser)
- die Objekterkennung (Hindernis, Person oder radioaktives Objekt)
- die Navigation (Umfahren von Hindernissen)
- die Kommunikation (zu Personen)
- die Objektbergung (radioaktive Gegenstände)

Zu jedem dieser Subsysteme wurden detaillierte Aktivitätsdiagramme erstellt. Diese sind unter <https://github.com/BrunoBerger/Rescue-Robot.gitim> Ordner "Diagramme/Subsysteme/..." zu finden.

Als erstes muss das *Firmengelände* definiert sein, welches in Abschnitt 4.2.2 näher beschrieben wird. Die *Fortbewegung* wird über die Ansteuerung der Motoren für Kettenantrieb (an Land) und Turbine und Ruder (im Wasser) realisiert (FR1 und FR2). Hierzu soll über den Wassersensor zunächst die Kapazität gemessen und bewertet werden. Gleichzeitig können über die Antennen am Rescue Robot Radiosignale empfangen werden. Die

Radiosignale enthalten eine ID und auch einen Standort (x- und y-Koordinate) zum Berechnen der Distanz. In Abb.7 wird die *Signalverfolgung* beschrieben (FR6). Zur Signalverfolgung wird die Antenne genutzt, welche die kürzere Distanz zum Radiosignal bzw. Funkturm hat. Erst wenn die Radiosignal Distanz < 1 ist, ist der jeweilige Funkturm erreicht.

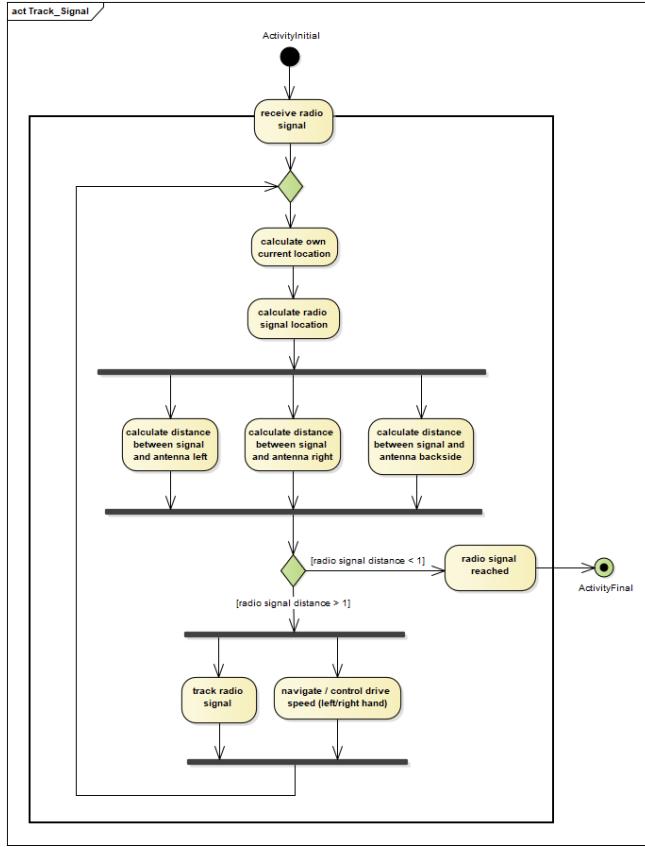


Figure 7. Subsystem: Aktivitätsdiagramm "Track Signal"

Zur *Objekterkennung* soll der Rescue Robot nicht nur den LIDAR-Sensor nutzen, sondern parallel sollen zur Person-Erkennung auch die Kamera Bilder ausgewertet werden. Wenn eine Person erkannt wurde, soll die *Kommunikation* gestartet werden. Dies passiert über Einschalten von Lautsprecher und Mikrophon. Der Rettungsdienst kann so mit der verletzten Person kommunizieren und über außen ggf. kleine "Erste Hilfe" leisten. Bei der *Objektbergung* kommt es zu mehreren Schritten, siehe Abb.8. Wird ein Objekt erkannt, wird der Greifarm in die berechnete Position gefahren. Der Greifer öffnet sich und mittels Geiger Sensor wird die Radioaktivität des Objekts gemessen. Soll das Objekt aufgrund seines Strahlungswerts eingesammelt werden, schließt der Greifer sich. Gleichzeitig sind die Stützen des Rescue Robots ausgefahren. Nun wird der Greifarm angehoben und das Objektgewicht wird mittels Kraft- und Druckaufnehmer gemessen. Wird das maximale zulässige Greif-Gewicht überschritten, öffnet der Greifer wieder und lässt das Objekt fallen. Ist das Objektgewicht im zulässigen

Bereich, fährt der Greifarm die Position über der Box an, der Deckel wird geöffnet und das Objekt wird in die Box gelegt. Die funktionale Anforderung FR3 ist somit erfüllt. Anschließend wird das aktuelle Transportgewicht berechnet. Ist dies größer als das maximal zulässige Transportgewicht, soll der Rescue Robot wieder zurück zu seinem Startpunkt auf dem Firmengelände fahren, damit die Box geleert werden kann.

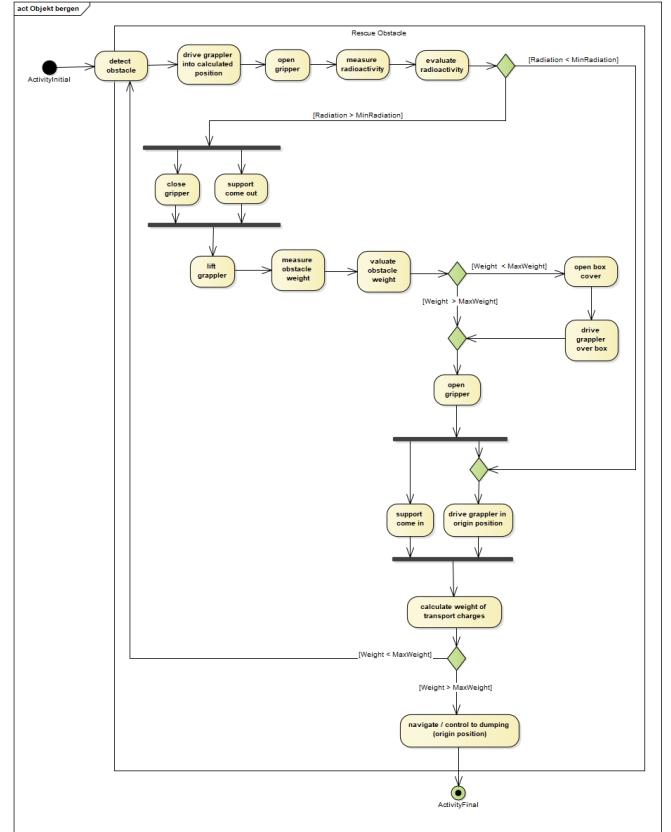


Figure 8. Subsystem: Aktivitätsdiagramm "Rescue Object"

## 4. Evaluation

In diesem Abschnitt werden die verschiedenen Implementationen vorgestellt, die durchgeführt wurden um jeweils verschiedene Anforderungen aus Abschnitt 1.3 zu erfüllen.

Berger

### 4.1. 3D Modell

Berger

Der grundsätzliche Aufbau des Roboters hat sich nach dem kombinierten Prototypen nicht stark verändert. Die Aufgabe des finalen Modells war es also genauer zu definieren aus welchen Bauteilen der Roboter besteht und zu beweisen das alle angedachten Funktionen im Gerät unterzubringen sind.

Wie in Abbildung 9 zu sehen, besteht der finale Roboter aus einer großen Grundplatte, die durch eine Rückwand und

einen Deckel abgedeckt wird. Dies vereinfacht die Wartung der inneren Komponenten und erlaubt das komplett ausbauen der beiden Container.(NFR9)

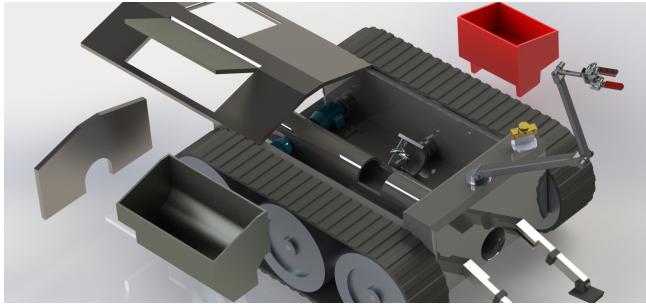


Figure 9. Modell final TEST

Wie in Abbildung 9 zu erkennen wurde der Greifarm des Roboters im Vergleich zum Prototypen deutlich verstärkt. Dies verbessert die Fähigkeit des Arms besonders schwere Objekte zu bergen.(FR3) Als Material für den Greifarm wurde vorerst Carbonfasern gewählt, gut zu sehen in Abbildung 11. Dies wurde gemacht um die Hebelwirkung des Arms zu verringern, wenn dieser sich auf weite Distanzen ausstreckt. Ob die Konstruktion aus dem Material stark genug ist um auch schwere Objekte zu bergen, muss noch getestet werden und könnte die Wahl des Materials noch verändern.

Die Ketten wurden um relativ sanfte Zähne erweitert um die Griff-Fähigkeit auf losem Untergrund zu verbessern.(FR1, NFR1) Sie werden angetrieben durch jeweils einen Elektromotor, der am hinteren Rad angebracht ist, besonders gut zu sehen in Abbildung 10.

In der Mitte der Grundplatte befindet sich eine Aussparung in der Wasser-Röhre. Dort wird der Propeller an einer Klappe befestigt, die wie in Abbildung 9 gezeigt aufgeklappt werden kann, zur Installation oder Wartung.(NFR9) Am Ende der Röhre befindet sich das Ruder, wie in Abbildung 10 zu sehen.(FR2, NFR2)

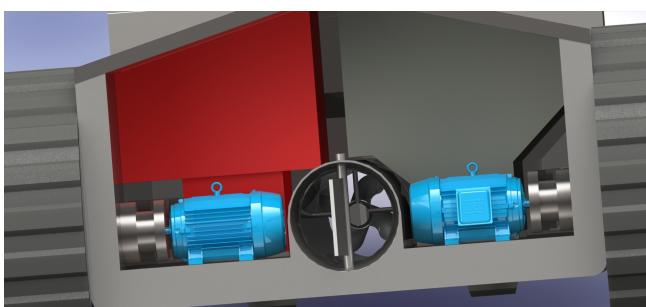


Figure 10. Modell back

Die beiden Container werden mit einfach von oben in den Roboter eingesetzt. Sie werden festgehalten von dem Deckel und den Seitenwänden, sowie von kleinen Erhebungen in der Grundplatte, die die Füße der Container umschließen. Die Klappe die den rechten Container verschließt ist in dem Deckel integriert.(FR4)

Die Stützen an der Vorderseite blieben im Ansatz gleich, wurden aber durch eine Gummi-Dichtung in der Mitte des Beins erweitert, zu sehen unten rechts in Abbildung 9. Sobald der Roboter Wasser erkennt, können die Stützen komplett eingefahren werden, wodurch die Dichtungen das sonst leicht durchlässige Loch im Chassis abdichten.

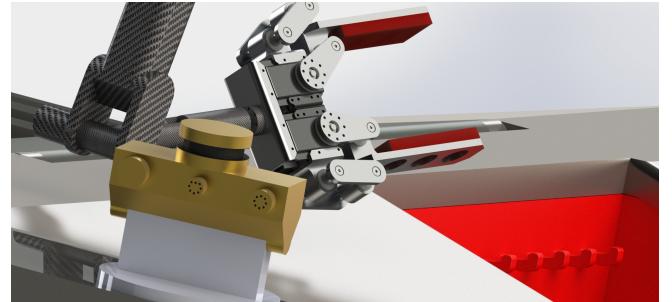


Figure 11. Werkzeuge des Roboters

Abbildung 11 zeigt deutlich den Aufbau von dem Sensor-Array und dem Greifer. Der Lautsprecher und die Kamera, Mikrofon und LIDAR Sensoren befinden sich auf einer kipp- und drehbaren Halterung. Dies ermöglicht die komplett Aufnahme der Umgebung.(NFR8)

Der Greifer ist angetrieben von 2 Motoren und setzt deren Drehbewegung in eine parallele Bewegung der Greifer-Platten in rot um. Dies ermöglicht sanftes und rutschfestes Greifen von viele unterschiedlichen Formen.(NFR3.1)

Rechts im Hintergrund von Abbildung 11 befindet sich der offene Container. Zu erkennen sind die vier Haken, an denen eventuelles Erste-Hilfe-Material oder ähnliches aufgehängt werden kann, um den Container aufgeräumt zu halten.(NFR5)

## 4.2. Implementierung in C#

*Walter*

Die Software wurde mithilfe von C# implementiert, aus dem Hintergrund, dass Unity diese Sprache verwendet. So kann das ganze Projekt in einer Sprache einheitlich implementiert werden.

C# eignet sich durch die Objekt-Orientierung sehr gut für die Beschreibung komplexer Systeme. Jedes Objekt, welches in dem behandelten Szenario vorhanden ist, wurde im Code implementiert. Die vorhandenen Klassen werden in einem Klassen-Diagramm in Abbildung 6 gezeigt.

Jede dieser Klassen hat verschiedene Methoden und Attribute. Die Objekte werden in Abschnitt 4.2.1 weiter erläutert.

**4.2.1. Objekte.** Die Simulation des Roboters, sollte so genau wie möglich sein, daher wurden alle Gegenstände, die mit dem Roboter interagieren als eigenständige Objekte angelegt. Bei den Objekten kann es sich, zum Beispiel, um sogenannte Ground-Objekte handeln. Diese Ground-Objekte beschreiben jedes Objekt, welches auf der Karte zu finden ist. So erben die Obstacle-Objekte von den Ground-Objekten

```
public abstract class Ground
{
    public int positionX;
    public int positionY;
    bool traversable;

    public Ground(int posx, int posy)
    {
        this.positionX = posx;
        this.positionY = posy;
    }
    public abstract bool returnTraversable();
}

public class Obstacle : Ground
{
    public Obstacle(double weight, double size, int posX, int posY) : base(posX, posY)
    {
        this.traversable = false;
        this.weightKG = weight;
        this.size = size;
    }

    Console.WriteLine("Obstacle generated: X: {0} , Y: {1} , Weight: {2} , Size: {3}")
}
```

Figure 12. Vererbung von Ground-Objekt zu Obstacle-Objekt

Zusätzlich zu den Ground-Objekten gibt es Peripherie-Geräte für den Roboter, wie zum Beispiel Kamera oder Mikrofon. Auch Bauteile wie Motoren wurde mit verschiedenen Eigenschaften implementiert. So kann ein Motor einen Namen, Anfangs- und End-position, Geschwindigkeit und State besitzen.

Die Rescue Bot Klasse, die in Abbildung 13 zu sehen ist, beinhaltet die Sensoren und Aktoren, welche durch den Roboter miteinander interagieren. Dazu zählen Kamera, Antennen, der LIDAR Sensor und Greifarm.

```

public class Rescuebot
{
    int positionX;
    int positionY;
    int startPositionX;
    int startPositionY;
    int maxTransportWeight = 200;
    double currentLoad = 0;
    // Bot sollte wissen was sich um ihn herum befindet und auf welchem untergrund er sich aktuell befindet

    Ground left;
    Ground right;
    Ground behind;
    Ground inFront;
    Ground current;

    Motor motorChainDriveLeft;
    Motor motorChainDriveRight;
    Motor turbine;
    Motor rudder;

    LeftAntenna leftAntenna;
    RightAntenna rightAntenna;
    BacksideAntenna backsideAntenna;

    Grappler grappler;
    Support support;

    CapacitySensor capacitySensor;

    LIDARSensor lidar;
    GeigerCounter geiger;
    Camera camera;
    Microphon microphone;
    Loudspeaker loudspeaker;

    Navigation navigation;
    BoxCover boxCover;
    Premises premises;
}

```

Figure 13. Rescue Bot Klasse

So befindet sich zum Beispiel der LIDAR Sensor, der Greifarm und der Geigerzähler innerhalb der Rescue Bot Klasse. Durch diese Verkettung kann der Rescue Bot auf die Methoden der Geräte, ebenfalls Klassen, zugreifen.

**4.2.2. Karte.** Die Karte wird aus einem zweidimensionalem Array generiert. Dieses Eingabe Array besteht dabei aus verschiedenen Strings. Aus dem Input wird mithilfe einer Switch-Case Anweisung ein neues Array generiert, welches aus Objekten besteht, die den Strings innerhalb des Eingabearrays entsprechen.

Arrays entsprechen. Das Eingabe Array wird in Abbildung 14 dargestellt.

Figure 14. Zweidiemensionales Array als Karte

So wird zum Beispiel aus einem "R" innerhalb des Eingabe Arrays ein radioaktives Objekt erstellt, mit zufälliger Größe, Gewicht und Strahlung als Attribute. Der Prozess der Objekt-Instanziierung wird in Abbildung 15 veranschaulicht.

Eine Aufschlüsselung der einzelnen Buchstaben und ihre Bedeutung wird in folgender Tabelle erläutert:

Zeichen	Bedeutung
0	Frei befahrbarer Untergrund
W	Wasser
X	Wand / Äußere Begrenzung
S	Startpunkt
F	Funkturm
R	Radioaktives Objekt
P	Zielperson

Die Objekte werden direkt in das objArr Array gespeichert, was beispielhaft in der vorletzten Zeile in Abbildung 14 zu sehen ist.

```

        case "R":
            // NextDouble = wert zwischen 0.0 und 1.0
            // randSizeRad = Wert zwischen 2 und 12
            // randWeightRad = Wert zwischen 3 und 23
            // randomRad = Wert zwischen 0 und 50
            // Runden auf 4 Nachkommastellen

            double randSizeRad = Math.Round(random.NextDouble()) * 10 + 2, 4, MidpointRounding.ToEven);
            double randWeightRad = Math.Round(random.NextDouble()) * 20 + 3, 4, MidpointRounding.ToEven);
            double randRad = Math.Round(random.NextDouble()) * 50 + 4, 4, MidpointRounding.ToEven);
            this.objArr[i,j] = new RadioactiveWebstable(randWeightRad,randSizeRad,j,i,randRad);

            break;
    }
}

```

Figure 15. Instanziierung von Radioaktiven Objekten

**4.2.3. Erkennung und Bergen von Radioaktiven Gegenständen.** Die Erkennung und das Bergen von radioaktiven Gegenständen ist eines der Hauptfeatures des Rescue Bots. Die Erkennung dieser Objekte erfolgt in mehreren Schritten. Bevor der Bot sich in Bewegung versetzt, wird über den LIDAR Sensor die unmittelbare Umgebung gescannt. Hierbei werden die Punkte vor, hinter, links, rechts und unter dem Bot erkannt. Felder, welche diagonal zum Bot liegen, werden ignoriert. Sollte sich auf einem Feld, das direkt an den Bot grenzt ein radioaktives Objekt liegen, wird dieses aufgesammelt, wenn es folgenden Kriterien entspricht: Größe, abgegebene radioaktive Strahlung und Gewicht. Der maximale Wert in allen drei Fällen beträgt 100 Einheiten.

Zuerst wird die Größe des Objektes mithilfe das LIDAR Sensors geprüft. Liegt das Objekt unterhalb der maximalen Größe, wird der Greifer in Richtung des Objektes bewegt und die radioaktive Strahlung durch den Geigerzähler gemessen. Sollte die Strahlung den bestimmten Wert übersteigen, wird das Objekt gegriffen und das Gewicht gemessen. Ist auch das Gewicht geringer als das maximale, kann der Gegenstand eingesammelt werden.

Dieser Prozess kann auch anhand des Aktivitätsdiagramms in Abbildung 8 nachvollzogen werden.

**4.2.4. Navigation.** Ziel ist es, den Bot autonom durch die generierte Karte fahren zu lassen. Um dies zu realisieren, beinhaltet die Karte drei Funktürme.

Jeder dieser Funktürme besitzt eine ID und Position auf der Karte. Diese Daten kann jeder der drei Funktürme an den Rescue Bot senden. Aus entsprechenden Daten kann der Bot entscheiden welcher Funkturm der nächste ist und in selbige Richtung fahren.

Die Berechnung der Distanz wird in Abbildung 16 gezeigt.

```
public double calcDist(int radioPosX, int radioPosY)
{
    int difX = this.positionX - radioPosX;
    int difY = this.positionY - radioPosY;
    if (difX < 0)
    {
        difX = difX * (-1);
    }
    if (difY < 0)
    {
        difY = difY * (-1);
    }
    double dist = Math.Sqrt(Math.Pow(difX, 2) + Math.Pow(difY, 2));
    if(dist == 1)
    {
        // wenn der Bot direkt an einem Turm steht ignoriert er die
        dist = 100;
    }
    return dist;
}
```

Figure 16. Berechnung der Distanz zwischen zwei Punkten auf einer Fläche durch den Satz des Pythagoras

Entspricht die gemessene Distanz einer Längeneinheit (dist = 1), steht der Bot direkt neben dem Funkturm. Um den Funkturm für weitere Berechnungen zu ignorieren, wird die Distanz auf eine hohe Zahl gesetzt, in Abbildung 16 ist dies im Beispiel 100. Das hat die Folge, dass nun die kürzeste Distanz zwischen den zwei verbleibenden Funktürmen gewählt wird.

Um sich zu den verschiedenen Funktürmen zu bewegen, können unterschiedliche Ansätze gewählt werden. Im Folgenden wird einer dieser Ansätze beschrieben.

In diesem Ansatz wird die Differenz der X- und Y-Koordinate berechnet und anschließend versucht diesen Differenzwert gegen Null zu bringen. Dies geschieht indem zwischen einer Bewegung in X-Richtung und in Y-Richtung gewechselt wird, um eine diagonale Bewegung zu ermöglichen. Auf diese Art und Weise kann die gefahrene Strecke minimiert werden.

Sollten sich Hindernisse auf dem Weg befinden, weicht der Bot in eine zufällige Richtung aus und versucht erneut

den Weg zum Ziel einzuschlagen. Dieser Vorgang wird so lange wiederholt, bis das Hindernis umfahren ist.

**4.2.5. Test Case.** Zum Testen des Systems wurde ein Szenario entwickelt, welches einige der Funktionen des Roboters überprüft.

So soll getestet werden ob der Roboter den Rotor einschaltet, wenn er schwimmt, ob er radioaktive Gegenstände eigenständig findet und einsammelt, und zum Schluss, ob eine Zielperson erkannt wird und mit ihr interagiert werden kann.

Folgende Einschränkungen wurden dabei festgelegt:

- Es gibt genau eine Zielperson, welche sich an einem der in Abschnitt 4.2.4 beschriebenen Funktürme befinden muss
- Die Radioaktiven Gegenstände dürfen frei verteilt werden
- Hindernisse dürfen frei verteilt werden
- Es wird auf Sicht einschränkungen durch Nebel verzichtet
- Bewegliche Hindernisse werden nicht berücksichtigt und entfernt

Das Ziel des Test-Case ist, den Roboter bis zum Erkennen einer Zielperson an den Funktürmen vorbeifahren zu lassen. Während dieser Suche, soll jedes radioaktive Objekt eingesammelt werden, solange der Bot nicht voll beladen ist. Zudem soll angezeigt werden, welcher Motor aktuell verwendet wird.

Der Programmablauf ist als erfolgreich anzusehen, wenn die Zielperson gefunden und die aktuelle radioaktive Ladung ausgegeben wurde.

```
Found Radioactive Object at X: 4 and Y: 2!
Support is extendet!
Opens Gripper!
Opens Gripper!
Closes Gripper!
Closes Gripper!
Box cover is open!
Moving Grappler to Box Position!
Opens Gripper!
Opens Gripper!
Grappler is in Parking Position
Box cover is closed!
RescueBot Load = 105,6293
Strong Ground Generated at X:4 and Y:2!
Support is retracted!
PosY1, PosX3
Person detected!
Found a Person at X: 2 Y: 1!!
Is everything alright?
Stopping the Program. Collected 105,6293Kg Radioactive Material!
```

Figure 17. Ausgabe des Programms (Einsammeln von Radioaktiven Gegenständen und Interaktion mit Zielpersonen)

```

Turning Motor Turbine on!
Turning Motor ChainDriveRight off!
Turning Motor ChainDriveLeft off!
PosY20, PosX8
PosY20, PosX9
Turning Motor ChainDriveRight on!
Turning Motor ChainDriveLeft on!
Turning Motor Turbine off!

```

Figure 18. Ausgabe des Programms (Verwendung des Aktuellen Motors)

Wie in Abbildung 17 abgelesen werden kann, findet der Rescue-Bot ein radioaktives Objekt, fährt den Greifer aus und entfernt es. Im weiteren Programmverlauf findet der Rescue-Bot die Zielperson und hat begonnen mit ihr zu interagieren. Somit ist die Funktion des Einsammelns von radioaktiven Gegenständen und der Interaktion mit Personen funktionsfähig.

In Abbildung 18 wird dargestellt welche Motoren aktuell verwendet werden und welche inaktiv sind. Fährt der Rescue Bot auf der Wasseroberfläche, so ist die Turbine in Betrieb, fährt er auf Land, sind beide Motoren für den Kettenantrieb aktiviert. Somit konnte auch diese Funktion validiert werden.

#### 4.3. Implementierung in Unity

Berger

Zusätzlich wurde der Wechsel der Antriebe des Roboters noch einmal genauer in Unity implementiert. Hierfür wurde das Modell aus Solidworks importiert und eine Beispiel Landschaft mit einem Wasser-Teich erstellt. Für die Kamera wurde eine Skript geschrieben, dass dem Benutzer erlaubt die Kamera um den Roboter zu rotieren.

#### 4.4. Bewegung über Rigidbodies

Die Bewegung des Roboters ist mit der Physik-Simulation von Unity realisiert. Dabei gibt es sogenannte "Rigidbody's", die einem Objekt zugewiesen werden können und Kollision und Gravitation umsetzen. Zusätzlich ist es möglich beliebige Kräfte auf die Rigidbody's einwirken zu lassen. Unser Rescue-Robot besteht in Unity aus einem Rigidbody für das Chassis, jeweils einem für jede Kette und einem für den Propeller. Sie sind mit sog. "fixed-joints" verbunden und sind wie in Abbildung 19 dargestellt positioniert. Dort sind zu Veranschaulichung farbige Sphären um den Schwerpunkt und Kraft-Ansatzpunkt der Rigidbody's gezeichnet.

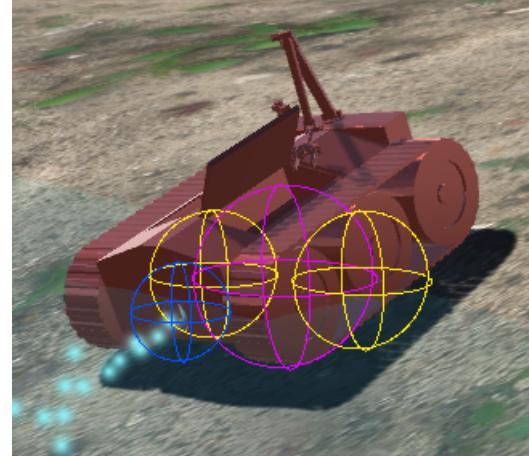


Figure 19. Roboter in Unity

Solange der Roboter Kontakt mit festem Untergrund erkennt, läuft die Fortbewegung über die Rigidbody's der Ketten. In diesem Modus kann der Roboter sich entweder geradeaus nach vorne und hinten bewegen, oder sich auf der Stelle drehen. Dies wird manuell vom Benutzer über die WASD-Tasten auf der Tastatur gesteuert. Die gewählte Richtung wird mit der Variable für die Kraft der Ketten multipliziert und als Kraft auf die Ketten angebracht. Da die Rigidbody's mit den fixed-joints verbunden sind, wird dadurch der gesamte Roboter bewegt.(NFR1)

Falls der Roboter Wasser berührt, läuft die Kraft stattdessen durch den Propeller.(FR7) Dabei wird ihm zusätzlich die Funktion des Ruders gegeben, das heißt falls zu Seite gesteuert wird, wirkt ein Drehmoment auf den Propeller und somit auf den gesamten Roboter.(NFR2)

Das schwimmen auf dem Wasser wurde deutlich vereinfacht umgesetzt. Ein halb transparenter Quader stellt das Wasser visuell dar und eine unsichtbare Ebene knapp unter der Oberfläche sorgt für die Kollision mit dem Roboter.

#### 4.5. Tests mit autonomen Fahren

Separat wurde das Skript autoDrive.cs erstellt. Dieses richtet den Roboter auf ein vorher zugewiesenes Objekt aus und lässt ihn in diese Richtung fahren. Beides findet auch hier physikalisch korrekt über Rigidbody's und einer festgelegten Motorkraft statt. Der Roboter überschießt dabei sowohl bei drehen als auch beim fahren sein Ziel nicht, sondern nähert sich erst immer schneller und immer langsamer an. Dieses Skript benutzt allerdings nicht den komplexeren Rigidbody-Aufbau aus Abschnitt 4.4, sondern wirkt alle Kräfte direkt auf den Haupt-Körper.

#### 4.6. Simulation des Greifarms

Berger

Die Aufgabe dieser Implementierung war es zu bestimmen in welcher Position die Gelenke eines Roboterarms seien müssen um mit der Spitze eine bestimmte Zielposition zu erreichen. Das Verfahren um dies zu berechnen nennt sich "Inverse Kinematik".

Für Python gibt es das ikpy-Package das genau dies umsetzt.<sup>1</sup> Für die Berechnung werden jedoch zuerst Informationen über den Roboterarm benötigt. Dies Beinhaltet die Ansatzpunkte der Arm-Elemente sowie die Art des Gelenke, zum Beispiel kippen oder drehen. Diese Daten könnten manuell erstellt werden, was viel Zeit in Anspruch nehmen würde. Deshalb wurde hierfür die Solidworks-Erweiterung URDF-Exporter benutzt.<sup>2</sup>

Diese ermöglicht es alle nötigen Informationen automatisch zu generieren. Dazu werden alle relevanten Bauteile des Modells ausgewählt und die Erweiterung erkennt nun die Rotationsachsen der Verbindungen und die Distanz und Position der Gelenke. Diese Daten werden dann im URDF als exportiert, was für "Unified Robot Description Format" steht eine weit verbreitete XML-Spezifikation für Roboter ist.[2]

Unser Python-Skript generiert dann mit Hilfe des ikpy Package aus den URDF-Daten eine sogenannte "Chain". Eine zweite Funktion nimmt diese Chain und ein Array mit Koordinaten entgegen und gibt die Ausrichtung der Kettenglieder aus. Diese ausgerichtete Kette kann dann, wie in Abbildung 20 dargestellt, mit dem Matplotlib-Package visualisiert werden.

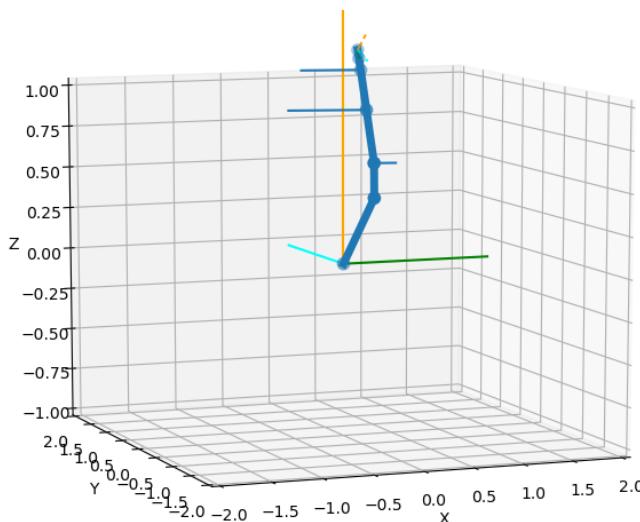


Figure 20. Matplotlib ouput

Alternativ könnten diese Daten auch an einen tatsächlichen Roboter-Arm weitergegeben werden. Der Rescue-Robot müsste nur die relative Position des zu greifenden Objekts berechnen und könnte dann das Objekt greifen und dann eine Position über dem Container anfahren.(FR3)

Zum jetzigen Zeitpunkt gibt es noch einige Probleme mit der Kette, die wahrscheinlich an einer falschen Anwendung der Solidworks-Erweiterung liegen.

1. <https://github.com/Phylliade/ikpy>

2. [http://wiki.ros.org/sw\\_urdf\\_exporter](http://wiki.ros.org/sw_urdf_exporter)

## 5. Zusammenfassung und Ausblick Berger

Zusammenfassend lässt sich sagen, dass alle Anforderungen aus Abschnitt 1.3 erfüllt wurden. Eine genauere Darstellung, von welchen Implementierungen die einzelnen Anforderungen erfüllt werden, lässt sich zusätzlich in der finalen Präsentation im GitHub Repository finden.<sup>3</sup>

Als Weiterführung könnten weitere Tests mit dem 3D Modell durchgeführt werden, um die Schwimmfähigkeit und die Effektivität des Propellers zu beweisen. Erste flow-simulation in SolidWorks haben bereits einen recht schnellen Fluss durch die Röhre des Roboters gezeigt. Zusätzlich könnte die Stabilität des Greifers unter Last geprüft werden.

Auch das Design könnte in Zukunft noch verfeinert werden, mit detaillierteren Hardware Schnittstellen und Protokollen für Randsituationen.

Die Software könnte noch um ein paar Features erweitert werden, wie zum Beispiel diagonales Fahren des Roboters oder Sicht einschränkungen wie Nebel.

## 6. Anhang

### 6.1. GitHub Übersicht

Berger

Zur Versionskontrolle wurde ein Git Repository genutzt, das auf GitHub gehostet wird. Darüber konnten wir gleichzeitig gemeinsam an dem Projekt arbeiten, was auch in sehr gleichen Anteilen stattgefunden hat. Die gesamte Arbeitszeit würden wir auf ungefähr 300 Stunden schätzen.

Zusätzlich wurden weitere Features von Github genutzt, wie die Project-Boards, die als Erweiterung des Trello-Boards genutzt wurden, um die Aufgaben für die jeweiligen Meilensteine festzuhalten.

Auch wurde GitHub-Actions genutzt, um "Nightly Builds" von der Implementierung in C# zu generieren. Dabei wird das Projekt auf einer virtuellen Windows Installation gebaut und dann der Status des Builds dann in der Readme der Repositorys angezeigt. Dies sieht aus wie in Abbildung 21, falls der Build Vorgang nicht fehlschlägt und informiert schnell über den aktuellen Stand des Projekts.



Figure 21. Build Badge

### 6.2. Source Code

Der Quellcode für alle Implementierungen ist auf GitHub zu finden.<sup>4</sup>

3. [https://github.com/BrunoBerger/Rescue-Robot/blob/master/Präsentation/Final\\_Presentation/Final\\_Project\\_RescueRobot.pptx](https://github.com/BrunoBerger/Rescue-Robot/blob/master/Präsentation/Final_Presentation/Final_Project_RescueRobot.pptx)

4. <https://github.com/BrunoBerger/Rescue-Robot/tree/master/Implementierung>

## **7. Affidavit**

We (Bruno Berger, Lukas Walter, Melanie Löbel) here-with declare that we have composed the present paper and work ourself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

## **References**

- [1] Andreas Birk and Stefano Carpin. “Rescue robotics—a crucial milestone on the road to autonomous systems”. In: *Advanced Robotics* 20.5 (2006), pp. 595–605.
- [2] *URDF Primer*. <https://de.mathworks.com/help/physmod/sm/ug/urdf-model-import.html>. Accessed: 2020-08-26. MathWorks, 2020.