CAT-MOUSE-CHEESE

Torben Miller (3164082)
Fabian Biedlingmaier (3224303)
Nicolas Hartlieb (3155952)

Heidelberg, am 31. August 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Aufgabenstellung

Bei dem „Cat-Mouse-Cheese“ Projekt gibt es zwei Roboter: eine Katze und eine Maus. Desweiteren befindet sich ein Stück Käse auf dem Spielfeld. Ziel der Katze ist es die Maus zu fangen und zu verhindern, dass die Maus den Käse bekommt. Die Maus muss dementsprechend der Katze ausweichen und zum Käse kommen. Das Spiel gilt als beendet, sobald:

a) Die Katze die Maus gefangen hat => Die Katze siegt

b) Die Maus kommt zum Käse => Die Maus siegt

c) Nach einer gewissen Zeit tritt weder a) noch b) ein => Unentschieden

In diesem Projekt wurde die Rolle der Katze übernommen.

1.1 Anforderungen

Bei den verwendeten Robotern handelt es sich jeweils um einen Turtlebot3 Burger.
Dieser Roboter hat eine maximale Lineargeschwindigkeit von 0.22 m/s und eine maximale WinkelgeschwindMatrikelnummerigkeit von 2.84 rad/s. [1]
Die Drehung des Roboters wird durch einen Radianten [2] angegeben. Ein positiver Wert führt zu einer Drehung gegen den Uhrzeigersinn, dementsprechend führt ein negativer Wert zu einer Drehung in Uhrzeigersinn. Um Hindernisse zu erkennen, besitzt er einen Laserscanner mit einer Scanreichweite von 120 bis 3500 mm und einer Auflösung von 1 Grad. [3] [4]

Für das Spiel gibt es jedoch Restriktionen der Roboter bezüglich Linear- und Winkelgeschwindigkeit.

Katze:

Lineargeschwindigkeit: konstant 0.22 m/s

Winkelgeschwindigkeit: -0.8 ... 0.8 rad/s

Maus:

Lineargeschwindigkeit: konstant 0.18 m/s

Winkelgeschwindigkeit: -2.84 .. 2.84 rad/s

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/#hardware-specifications
[2] Wertebereich [0:2 π]
[3] 1 Gard 0,0174533
[4] https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

1.1.1 Karten

Für das Projekt wurden seches Karten bereitgestellt. Drei für das Spiel und drei weitere Karten um Grenzfälle zu testen. Die drei Grenzfallkarten wurden bei der Entwicklung nicht berücksichtigt daher werden sie im folgenden nicht behandelt.

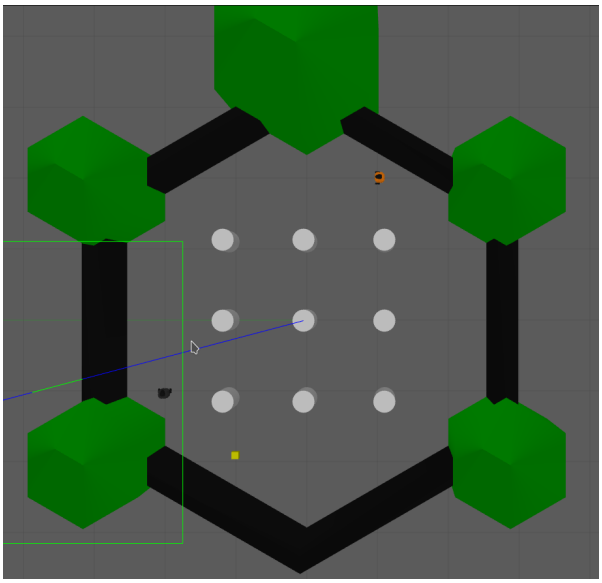

Abbildung 1: Welt 1

Die erste Welt (1) ist der stilistischen Sicht einer Schildkröte nachempfunden. Die Welt ist durch eine sechsseitige Mauer eingegrenzt Jede Ecke ist durch eine Säule abgerundet. In der Mitte der Welt befinden sich neun Säulen, die in einem Raster angeordnet sind. Der Käse ist an der gegenüberliegenden Seite zur Maus positioniert. Die Katze befindet nahe dem Käse leicht seitlich zwischen Maus und Käse.

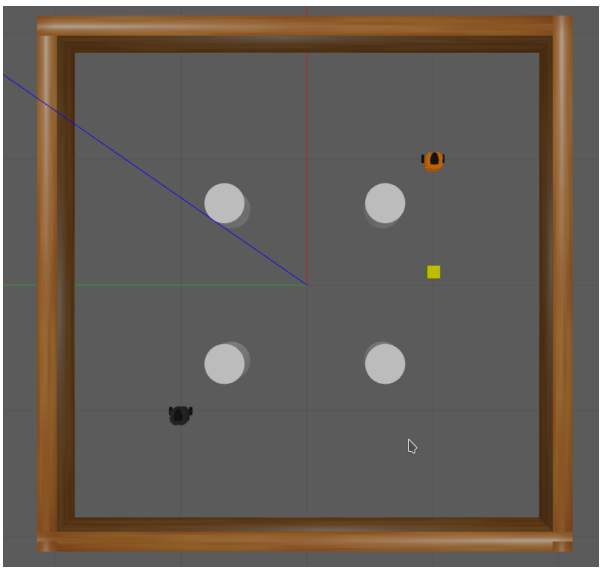

Abbildung 2: Welt 2

Die zweite Welt (2) ist quadratisch eingemauert und besitzt vier Säulen in der Kartenmitte. Katze und Maus sind an gegenüberliegenden Ecken positioniert. Der Käse liegt in der Nähe der Maus allerdings blickt die Maus in entgegengesetzte Richtung.

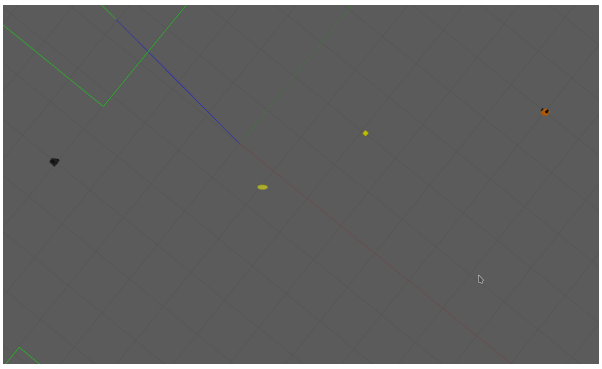

Abbildung 3: Welt 3

In der dritten Welt (3) befinden sich nur die Katze, die Maus und der Käse. Alle drei befinden sich auf einer Geraden.

1.2 Mögliche Implementierungsmöglichkeiten

Im Laufe des Semester wurden verschiedene Methoden, das Verhalten des Roboters zu implementieren, in der Vorlesung vorgestellt.

1.2.1 Kraftbasierte Kollisionsvermeidung und Homing

Bei diesen Verfahren wird bei der Kollisionsvermeidung eine Kraft berechnet, die von den Hindernissen auf den Roboter wirken. Je näher der Roboter an einem Hindernis ist, desto mehr wird er von diesem abstoßen. Beim Homing erfährt der Roboter eine Anziehungskraft auf einen Punkt im Spielfeld. Durch das Aufaddieren beider Kräfte, dem Homing und der Kollisionsvermeidung, ergibt sich die endgültige Kraft, die den Roboter von Hindernissen abstößt ihn jedoch gleichzeitig immer näher zum Ziel treibt.

1.2.2 Sense Plan Act

Hier wird für jede Folge von möglichen Zuständen der maximale Nutzen berechnet, der Roboter arbeitet somit vorausschauend. Dabei entsteht ein Suchbaum. Dieser Suchbaum kann mit dem MinMax-Algorithmus optimiert werden.

1.2.3 Optimal Control

Während bei MinMax die addierten Kosten minimiert oder der addierte Reward maximiert wird, werden bei optimal control die integrierten Kosten minimiert.

1.2.4 Path Planning

Durch eine gegebene Umgebungskarte versucht der Roboter den kürzesten Weg zu seinem Ziel zu berechnen. Um dies zu erreichen wird die Karte beispielsweise mit einem Occupancy grid in Zellen zerlegt und mit einem Shortest Path Algorithmus wie dem A*-Algorithmus die kürzeste Distanz vom Start zum Ziel berechnet.

1.3 Zielsetzung

Im Rahmen dieses Projekts wird ein kraftbasierter Ansatz implementiert. Im Laufe des Projekts wurden die Parameter so angepasst, dass sie möglichst optimal im Wettkampf gegen die Ansätze der anderen Teilnehmer des Projektes abschneiden.

2 Konzept

2.1 Grundverhalten

Die Katze hat zwei wesentliche Grundbestandteile. Sie kann ‚auf einen gegebenen Punkt, „homen“ und sie besitzt eine Kollisionsvermeidung.

2.2 Homing

2.2.1 Ziele

Für das Homing gibt es vier verschiedene Ziele (4).

1. der Käse
2. der Mittelpunkt zwischen Käse und Maus
3. der vermutete Punkt auf den sich die Maus zubewegt
4. die Maus

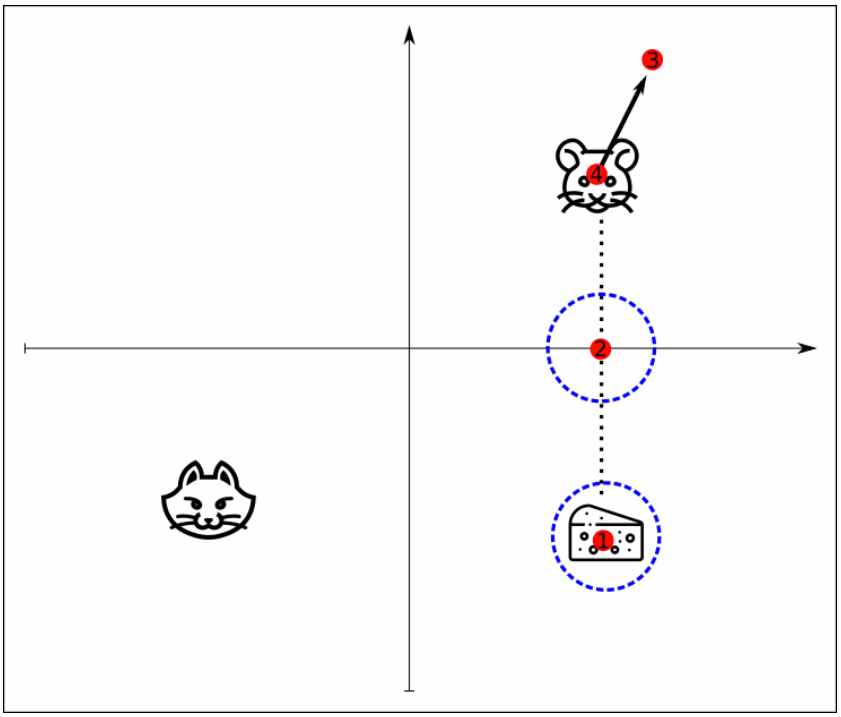

Abbildung 4: Ziele für das Homing

2.2.2 Zustandsautomat

Die Zielauswahl wird durch einen Zustandsautomaten (5) festgelegt. Der Zustandsautomat besitzt drei Zustände, mit dem Zustand „go to cheese“ als Startzustand. Im Zustand „go to cheese“ ist der Käse das Ziel, im Zustand „go to mid“ ist der Mittelpunkt zwischen Maus und Käse das Ziel und der Zustand „hunt“ fasst die zwei Ziele Maus und den vermutete Punkt auf den sich die Maus zubewegt zusammen. Wie der Zustand „hunt“ sich verhält wird im Abschnitt 2.2.3 genauer erläutert.

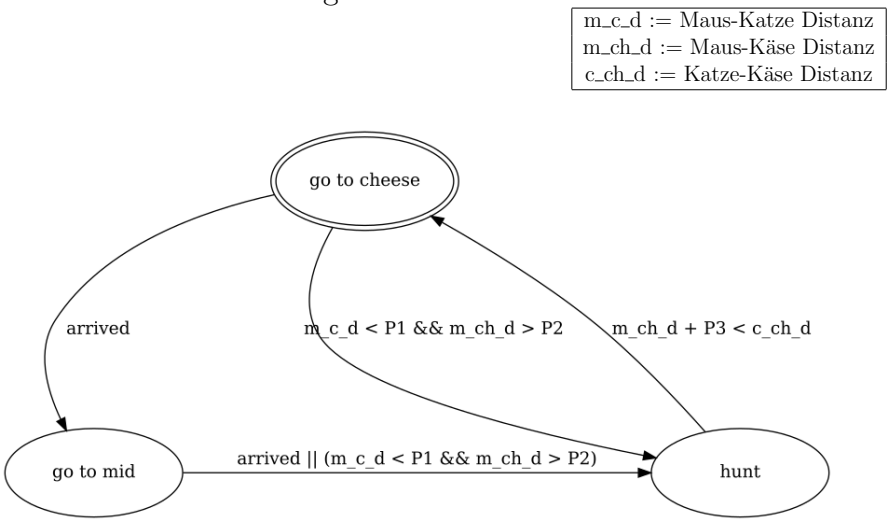

Abbildung 5: Zustandsautomat

Es gibt drei Kriterien die einen Zustandswechsel zur Folge haben. Das „arrived“ Kriterium ist hierbei das einfachste und wird erreicht sobald ein gewisser Abstand zum Zielpunkt unterschritten wird.
Das zweite Kriterium „$m\_c\_d < P_1 \wedge m\_ch\_d > P_2$“ besagt: sollte der Abstand zwischen Katze und Maus einen gewissen Wert P_1 unterschritten haben und zusätzlich der Abstand zwischen Käse und Maus einen weiteren Wert P_2 überschritten haben so wird der Zustand gewechselt. Umgangssprachlich ausgedrückt bedeutet es, sollte sich die Katze nah an der Maus befinden und die Maus aber noch weit genug vom Käse weg sein so wird die Maus gejagt. Wobei der Parameter P_1 „nah “ und der Parameter P_2 „noch weit genug“ definiert. P_1 und P_2 wurden empirisch festgelegt.
Das dritte Kriterum „$m\_c\_d + P_3 < c\_ch\_d$“ bedeutet: ist der Abstand zwischen der Maus und dem Käse kleiner dem Abstand zwischen der Katze und dem Käse so erfolgt der Zustandswechsel. Hier wird ein Wert P_3 hinzugefügt der als Hysterese fungiert.
Die Parameter P_1, P_2 und P_3 wurden empirisch festgelegt.

Im Zustand „go to cheese“ löst das „arrived“ Kriterium einen Wechsel in den Zustand „go to mid“ aus. Äquivalent dazu löst das gleiche Kriterium im Zustand „go to mid“ den Wechsel in den Zustand „hunt“ aus. Von beiden Zuständen, „go to cheese“ und „go to mid“, findet ein Wechsel in den Zustand „hunt“ statt, sollte das zweite Kriterium, „$m\_c\_d < P_1 \wedge m\_ch\_d > P_2$“, erfüllt sein. Im Zustand „hunt“ greift das dritte Kriterium, „$m\_c\_d + P_3 < c\_ch\_d$“, um den Wechsel in den Zustand „go to cheese“ einzuleiten.

2.2.3 Zustand: „hunt“

Im Zustand „hunt“ wird entschieden ob die Maus direkt als Ziel angefahren wird oder ob die Katze die Bewegung der Maus vorausberechnet. Im Fall der Vorausberechnung wird auch entschieden wie weit vorausberechnet wird.
Die Berechnung der Bewegung bezweckt eine sich vom Käse entfernende Schlingerbewegung der Maus (6). Dieser Ansatz fungiert als Gegenspieler zu einem kraftbasierten Ansatz der Maus.

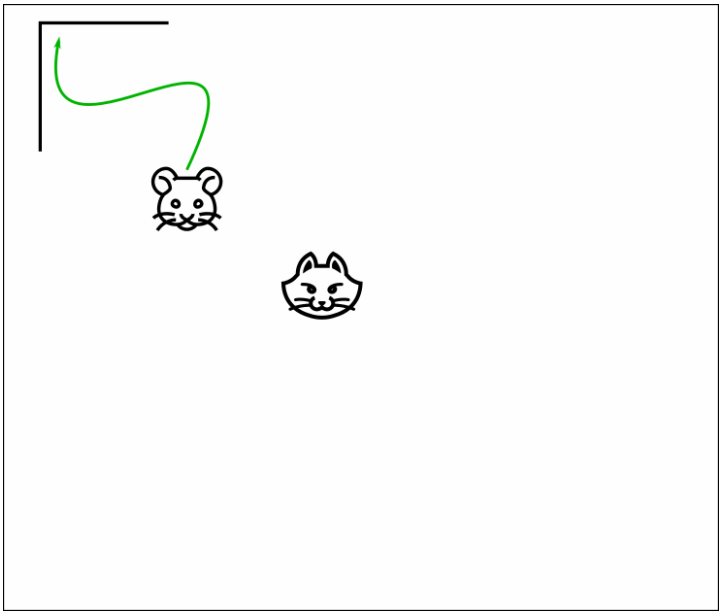

Abbildung 6: Schlingerbewegung

Um diese Verhalten zu erreichen wird die Bewegung der Maus verschieden weit vorausberechnet. Im Zustand hunt “ wird, bis zu einem gewissen Schwellwert, nicht direkt die Maus angefahren sondern ein vorausberechneter Punkt vor der Maus. Je näher die Katze der Maus kommt desto weiter steuert sie vor die Maus. Unterschreitet die Distanz zwischen Katze und Maus einen gewissen Schwellwert, so wird direkt die Maus als Ziel gesetzt.

Den Wert der Skalierung der Vorausberechnung wird r-Wert genannt. Er wird anhand einer Funktion berechnet. Gegeben aus den Anforderungen benötigt besagte Funktion eine sägezahnähnliche Form.
Hierzu wurde die Funktion $r = \frac{2}{x} - \frac{2}{x^3}$ gewählt mit $x := Distanz(Maus, Katze)$ (7).

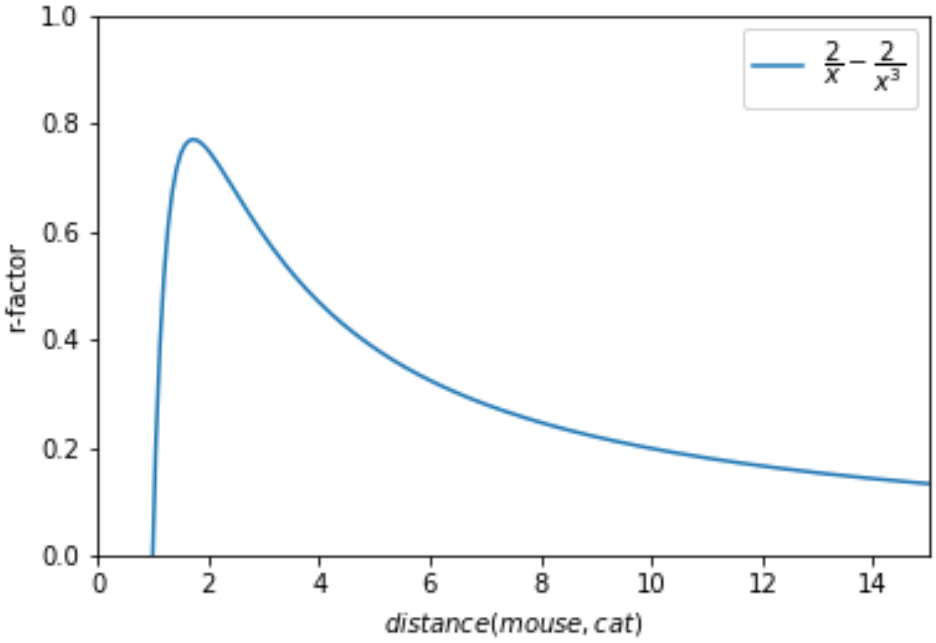

Abbildung 7: r-Faktor Funktion

2.3 Collison Avoidance

Bei der Kollisionsvermeidung werden nicht alle 360 Sensorwerte berücksichtigt. Der relevante Bereich wird durch einen Parameter [5] angegeben. Objekte die im Scanbereich sind haben eine abstoßende Wirkung auf die Katze.

2.3.1 Funktionsentwicklung

Folgende Kriterien sind für die Funktion der Kollisionsvermeidung wichtig.

- Je geringer die Distanz zu einem Objekt [6] desto größe die Kraft
- Je geringer der Winkel eines Objekts zur Katze, desto größer deren Kraft [7]

Nun ist ein genauer Blick auf die Wertebereiche der relevanten Variablen nötig.

Variable	Wert	Auswirkung auf die Kraft
sensor.angular	$[0 : rel\_phi]$	negativ (Drehung mit den Uhrzeigersinn)
sensor.angular	$[2\pi : 2\pi - rel\_phi]$	positiv (Drehung gegen den Uhrzeigersinn)
sensor.range	gegen 0,8	Kraft gegen 0
sensor.range	gegen 0	Kraft wird größer

Folgende Funktion erfüllt die aufgestellten Kriterien und weist die gewünschten Verhaltensweise auf.

$$Force = -\sin(sensor.angular) * (rel\_range - sensor.range)$$

Die negative Sinusfunktion wird hier als Grundgerüst genommen, sa sich sensor.angle zwischen 0 und 2π bewegt. Werte nahe der 0 führen zu einer negativen Kraft und bewirken somit eine Rechtsdrehung. Werte nahe 2π bewirken hingegen eine Linksdrehung. Die so entstehende winkelabhängige Kraft wird nun mit der Distanz multipliziert, von der vorher von der Maximaldistanz abgezogen wird. Somit ist der Multiplikator größer, je kleiner die Distanz ist.

2.3.2 Sonderfälle

Die Maus wird von den Sensoren erfasst und muss daher aus der Kollisionsvermeidung herausgerechnet werden.
Um dieses Problem zu lösen wurde ein Spezialfall entwickelt. Ziel ist es alle Sensorwerte, die die Maus erfassen zu bestimmen und diese in der Kollisionsvermeidung nicht zu berücksichtigen.

[5] rel_phi
[6] sensor.range
[7] Ein Objekt in Fahrtrichtung soll eine größere Kraft haben als ein Objekt welches sich Seitlich befindet

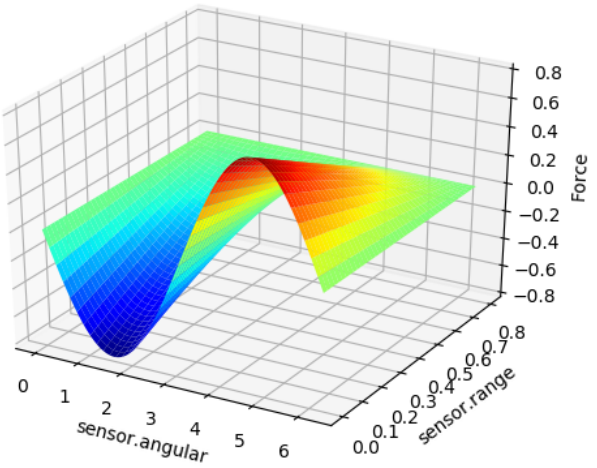

Abbildung 8: Funktion Collison Avoidance

Strategie:

1. Maus ist in Scanreichweite
2. Gescannte Range entspricht der Entfernung zur Maus
3. Relevante Winkle in Bezug auf die Größe der Maus berechnen
4. Alle sensor.range-Werte der relevanten Winkel auf Infinity setzen

Sobald die Maus in Sensorreichweite ist, ist es wichtig zu überprüfen ob die Sensor.range der mathematischen Distanz zur Maus entspricht. Falls man dies nicht überprüft kann es passieren, dass sich ein Hindernis zwischen Maus und Katze befinden. Das Hindernis wird von dem Algorithmus als Maus tituliert und für die Kollisionsvermeidung nicht berücksichtigt. Dies hat zur Folge, dass die Katze in das Hindernis hineinfährt.
Sofern diese Fälle berücksichtigt worden sind, kann es daran gehen die relevanten Sensorwerten zu löschen, dies bedeutet sie auf Infinity zu setzen.
Zuerst wird der angle_mouse berechnet dies ist der Winkel aus Sicht der Katze zur Maus, mit den globalen Positionsangabne[8].

$$angle\_mouse = \tan(\frac{\Delta x}{\Delta y}) - cat\_phi$$

[8]Positionen der Maus in odom Sysem

Angle_mouse gibt den Winkel der Maus zur Basis der Katze an, nun wird berechnet welcher Sensorwinkel auf die Mitte der Maus zeigt:

$$laserMouse\_middle = |\frac{(angle\_mouse + 2\pi)\%(2\pi)}{angle\_increment}| * angle\_increment$$

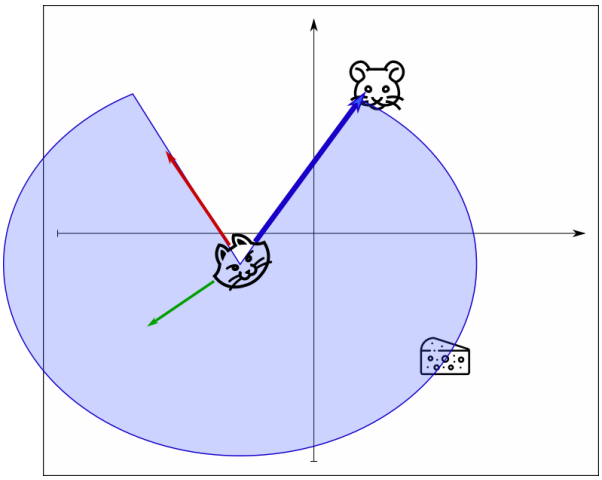

Abbildung 9: Sensorwinkel Katze

Nachdem der Sensorwinkel zu Mitte der Maus berechnet wurde, wird mit Hilfe des Radiuses(mouse_radius) der Maus die Außenwinkel berechnet.

$$laserMous\_border = |\tan(\frac{mouse\_radius}{\Delta cat\_mouse})/angle\_increment| * angle\_increment$$

Winkel zwischen Rot & Lila:= $\tan(\frac{mouse\_radius}{\Delta cat\_mouse})$
Blau := mouse_radiu s
Rot := Δcat_mouse

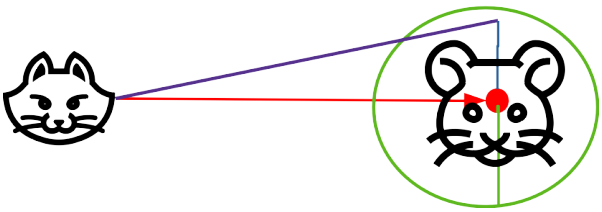

Abbildung 10: Beziehungen Katze zu Maus

Nachdem berechnet wurde welche sensor.angle-Werte momentan die Maus scannen, werden alle Sensor.range-Werte dieser Winkel auf Infinity gesetzt. Nachdem alle relevanten Werte des Sensors auf Infinity gesetzt sind, ist die Maus für die Kollisionsvermeidung

unsichtbar und die Katze sieht die Maus nicht als Hindernis an, welche ihre Fahrtrichtung beeinflusst.

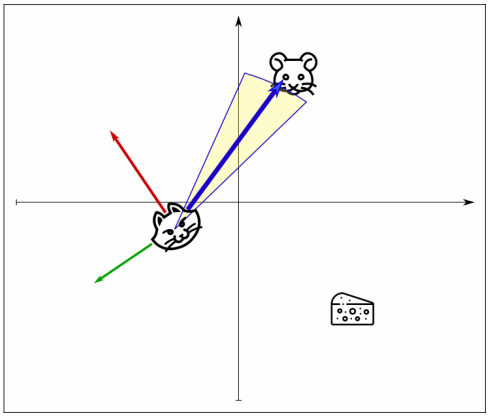

Abbildung 11: Lasersensor ignorieren

Ein weiterer Spezialfall ist der Käse. Er ist zu niedrig um vom Lasersensor erfasst zu werden.

Strategie:

1. Käse ist in Scanreichweite
2. Relevante Winkle in Bezug auf die Größe und Position des Käses berechnen
3. Alle sensor.range-Werte der relevanten Winkel Werte zuweisen

Mit dem Globalen Koordinaten wird überprüft ob der Käse theoretisch in Sensorreichweite ist. Parallel zum Vorgehen bei der Maus wird der Winkel vom Käse zur Katze und der Sensorwinkle zur Mitte des Käses berechnet.

$$angle\_cheese = \tan(\frac{\Delta x}{\Delta y}) - cat\_phi$$

$$laserMouse\_cheese = |\frac{(angle\_cheese + 2\pi)\%(2\pi)}{angle\_increment}| * angle\_increment$$

Nachdem berechnet wurde wo der Käse sich befindet, werden mithilfe der Kreisfunktion die sensor.range-Werte berechnet und gesetzt. $sensor.range = \sqrt{rad^2 - x^2}$

$$sensor.range_i = \Delta cheese - \sqrt{cheese\_rad^2 - \frac{\tan(laser\_cheesMiddl - angle_i)^2}{\Delta cheese}}$$

Blau := $\frac{\tan(laser\_cheesMiddl - angle_i)}{\Delta cheese}$
Rot := Δcat_cheese
Lilia := $sensor.range_i$

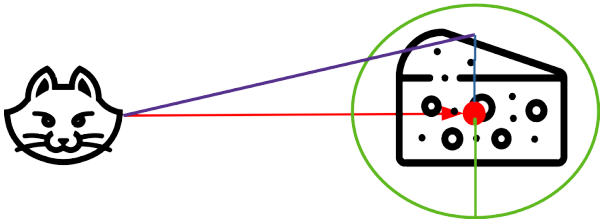

Abbildung 12: Überblick Katze Käse

Nachdem alle Werte berechnet sind kann die normale Kollisionsvermeidung durchgeführt werden. Die Katze registriert nun dort wo sich der Käse befindet einen Halbkreis und weicht diesem Objekt aus. Es wird der Katze eine Säule an der Position des Käse vorgetäuscht.

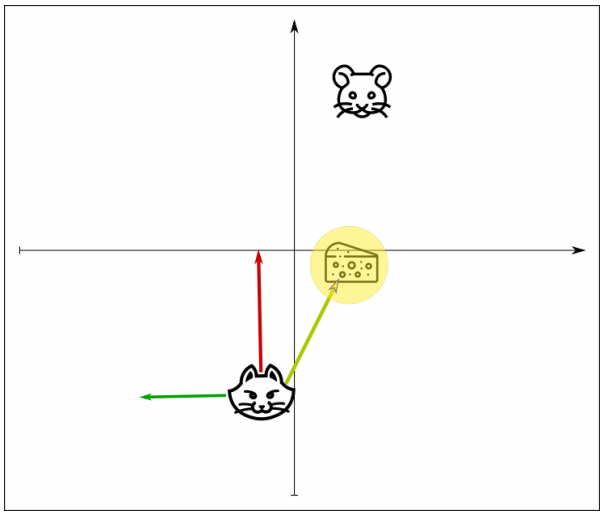

Abbildung 13: Käse als Hindernis Simuliert

3 Diskussion und Ausblick

3.1 Bewertung

Der kraftbasierte Ansatz erweist sich nach der Auswertung auf den Welten eins bis drei als praktikabel. Durch seinen rein reaktiven Charakter ist er weniger rechenintensiv als pfadplanende Ansätze. Der Erfolg ist allerdings sehr abhängig von den Karten und den implementierten Verhalten der Gegenspieler. Für komplexerer Karten, wie Szenario A-C, wäre ein Planungsalgorithmus vorteilhafter. Bei der Sichtung der Videos ist desweiteren aufgefallen, dass keine der Gegenspieler ihre volle Drehgeschwindigkeit nutzen und dadurch möglicherweise Potenzial verschenken

3.1.1 Fehlerhafter Zustandsautomat

Bei der Auswertung des Videos c_1_15_1_c ist zu sehen, dass der Zustandsautomat nicht differenziert genug ist. In dem Video sieht man, wie die Katze aus dem Zustand „hunt“ in den Zustand „go to cheese“ wechselt, da die Maus näher am Käse ist als die Katze. Nachdem die Katze nun zum Käse fährt und diesen erreicht wechselt die Katze wieder in den Zustand „hunt “. Zu diesem Zeitpunkt ist die Maus jedoch schon zu nahe am Käse und die Drehgeschwindigkeit der Katze reicht nicht aus um die Maus noch zu erreichen. Das Problem ließe sich auf verschiedene Weise lösen, zeigt jedoch das grundsätzliche Problem, dass der Zustandsautomat nicht genügend Grenzfälle abfängt. Ein Ansatz wäre, vor dem Wechsel vom Zustand „hunt “ in den Zustand „go to cheese“ den Abstand von Maus zur Katze mit der Distanz von Maus zum Käse zu vergleichen.

3.1.2 Recovery Strategie

Bei Karten mit komplexen, polygonen Hindernissen oder Sackgassen (siehe Szenario A) ist die Katze zum jetzigen Zeitpunkt nicht in der Lage darauf zu reagieren. Eine Recovery Strategie für Sackgassen fehlt komplett. Eine Strategie ließe sich mittels neuer Zustände oder äquivalent mit einer Verhaltensfusion integrieren. Für komplexe, polygone Hindernisse wäre ein Path Planning Ansatz vorteilhafter.

3.1.3 Sinusfunktion

Bei näherer Betrachtung der Funktion zur Berechnung der Kraft (2.3.1) bei der Kollisionsvermeidung fällt auf, dass bei Winkelwerten gegen 0 rad und gegen $\pi/2$ rad die Funktion gegen 0 Rad geht. Dies hat zur Folge, dass Objekte nahe der Fahrtrichtung wenig bis keine Abstoßungskraft haben. Objekte die bei einem Winkel von $\frac{2}{\pi}$ rad und $3\frac{\pi}{2}$ rad liegen haben dagegen die Maximale Abstoßungskraft(Extrempunkte). Lösen ließe sich dieses Problem durch die Streckung der Funktion in x-Richtung, damit die Extrempunkte bei 0 und $\frac{2}{\pi}$ liegen.

3.2 Ausblick und Verbesserungen

Beim kraftbasierten Ansatz ergeben sich sehr viele Parameter. Die Optimierung dieser ist von elementarer Bedeutung für den Erfolg des Ansatzes. Die Parameter könnten mit Hilfe eines dafür konstruierten Neuronalen Netzes effektiver optimiert werden. Hierzu müsste jedoch eine eigene Simulation implementiert werden, da Gazebo hierfür zu langsam ist.
Des Weiteren könnte bei der Implementierung zur Vereinfachung der Zustandsautomat in eine Verhaltensfusion übertragen werden. Somit könnten die einzelnen Verhalten als eigenständige Ros nodes implementiert werden. Dadurch könnte man die Verhaltensweisen einzeln optimieren und situationsbedingt gewichten.