Qualitätssicherung

Fennec Racer

Autor: Team Fennec Bot
Letzte Änderung: 21.07.2021
Dateiname: Qualitätssicherung_Fennec_Bot.docx

Version: 1.0

Inhaltsverzeichnis

Copyright

Version Historie

Version:	*Datum:*	*Verantwortlich*	*Änderung*
1.0	30.06.21	Lukas Evers, Umut Uzunoglu, Hien Ahn Nguyen Manh, Son Khue Nguyen	

Vorhandene Dokumente

Alle für die vorliegende Spezifikation ergänzenden Unterlagen müssen hier aufgeführt werden

Dokument	**Autor**	**Datum**
Lastenheft_Fennec_Bot.pdf	Team Fennec Bot	02.05.21
Pflichtenheft_FennecBot.docx.pdf	Team Fennec Bot	19.05.21
Technische_Spezifikation_Fennec_Bot.pdf	Team Fennec Bot	16.06.21
Qualitätssicherung_Fennec_Bot.docx		

1 Testplan

Test-Objekt	Qualitätskriterien	QS-Teststufe 1 "Source Code, Komponente, Funktion"			Bemerkungen
		Test-Verfahren	Zyklus	Zuständig	
Dokumentation					
Source code	Verständlichkeit Lesbarkeit, Funktionale Vollständigkeit und Korrektheit	Editorial Review Technisches Review Gegenlesen	nach jeder Änderung, Meilenstein, am Ende	Teammitglied, Anwender	Der Source-Code muss verständlich und strukturiert sein.
Source code-Dokumentation	Verständlichkeit Lesbarkeit, Funktionale Vollständigkeit und Korrektheit t	Editorial Review Technisches Review Gegenlesen	nach jeder Änderung, Meilenstein, am Ende	Teammitglied, Anwender	Die Dokumentation des Source-Codes muss ausführlich durchgeführt sein.
…					
Applikation					
Funktionalitäten					
Umgebung scannen	Richtigkeit, Zuverlässigkeit	Funktionstest, Datentest, Lasttest	am Ende	Teammitglied	Es wird eine OGM (Occupany Grid Map) erstellt
Ball autonom suchen und erkennen	Richtigkeit, Robustheit, Zuverlässigkeit	Funktionstest, Datentest, Performanztest	am Ende	Teammitglied	Der Ball ist ungefähr fußballgroß.

Automatisch abschalten	Zuverlässigkeit	Lasttest	nach jeder Änderung	Teammitglied	Bei Überhitzung muss der Schutz der Hardware gewährleistet sein
Tore erkennen und auseinanderhalten	Richtigkeit, Zuverlässigkeit, Funktionalität	Funktionstest, Datentest	am Ende	Teammitglied	Es soll nur auf das richtige Tor geschossen werden.
Fernsteuerung	Ergonomie, Zuverlässigkeit, Benutzbarkeit	Funktionstest, Ergonomietest	am Ende	Kunde, Teammitglied	Bedienung soll leicht und intuitiv sein
nicht funktionale Eigenschaften / Anforderungen					
Hardware ist durch Chassis vor Schäden geschützt	Robustheit	Stresstest	nach jeder Änderung	Teammitglied, Anwender	
Inbetriebnahme des Roboters ist leicht und schnell	Benutzbarkeit	Lasttest	nach jeder Änderung	Teammitglied, Anwender	Unter zwei Minuten soll der Roboter startbar sein.
Wartbarkeit des Roboters	Wartung	Ergonomietest, Funktionsst	nach jeder Änderung	Teammitglied	Ladebuchse muss zugänglich bleiben.

Test-Objekt	Qualitätskriterien	QS-Teststufe 2 "Integration / Systemtest"			Bemerkung?
		Test-Verfahren	Zyklus	Zuständig	
Funktionalitäten					
Gegen Ball fahren und schießen	Funktionalität	Funktionstest, Zuverlässigkeit, Robustheit	am Ende	Teammitglied	Bilderkennung, Lokalisierung, Navigation und Hardware Controller spielen hier zusammen
Hindernisse umfahren	Richtigkeit, Zuverlässigkeit, Effizienz	Funktionstest, Performanztest	am Ende	Teammitglied	Navigation und die Integration des Ackerman Controllers sind hier zu testen, insbesondere auch die Recovery Behaviours bei Fehlern der Sensorik/Navigation
Sensorfusion	Funktionalität, Zuverlässigkeit	Lasttest, Funktionstest	am Ende	Teammitglied	Odometrie, IMU, Laserscan und vSlam können gemeinsam zur Verbesserung der Karten und der Navigation dienen
Sensorik einbinden	Funktionalität	Funktionstest	Meilenstein	Teammitglied	Integration von Komponenten in ROS

2 Testfälle

2.1 Einbindung RPLIDAR mit ROS

Pro Testfall soll das folgende Template angewandt werden:

Testfall	**Beschreibung**
Testfall-Nummer	01
Testart	Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	Laserscanner
Testziel	Man kann den Laserscanner mit ROS starten und die Sensorinformationen empfangen und zur Überprüfung anzeigen lassen
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit USB Schnittstelle hergestellt ● RPLiDAR ROS Package installiert ● Workspace und ROS Installation gesourced
Testfalldaten	● Starten des Lasers mit roslaunch rplidar_ros view_rplidar.launch
Erwartetes Verhalten	● In Rviz sollte man rote Punkte im Raum sehen welche die einzelnen Entfernungsmessungen des Lasers darstellen.

Testergebnis
Folgendes Template soll das Testergebnis jedes einzelnen Testfalls dokumentieren:

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwer[1]
Bemerkung			
Tester Kunde	Tester Auftragnehmer	Datum	
Umut Uzunoglu	Umut Uzunoglu	21.07.21	

[1] Die Beschreibung der Fehlerkategorien entnehmen Sie bitte dem beigefügten Anhang

2.2 Einbindung Intel Real Sense mit ROS

Testfall	**Beschreibung**
Testfall-Nummer	02
Testart	Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	Intel RealSense Kamera
Testziel	Die Kamera kann mit ROS gestartet werden und Kamerabilder sowie die Sensorinformationen werden empfangen und zur Überprüfung werden diese visuell dargestellt
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit USB Schnittstelle hergestellt ● RealSense-ROS Package installiert ● Workspace und ROS Installation gesourced
Testfalldaten	● Starten des Lasers mit “ roslaunch realsense2_camera rs_camera.launch”
Erwartetes Verhalten	● In Rviz kann man das Live-Bild der Kamera sehen ● In Rviz kann man die Tiefendaten der Kamera anzeigen lassen

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Umut Uzunoglu	Tester Auftragnehmer Umut Uzunoglu	Datum 21.07.21	

2.3 Verwendung eines Hardware Controllers für Ackerman Steuerung

Testfall	**Beschreibung**
Testfall-Nummer	● 03
Testart	● Integrationstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Jetracer
Testziel	● Es ist möglich Twist Message in ROS zu publishen und diese werden dann vom Hardware Controller in die einzelnen Bestandteile zerlegt um die Lenk- und Antriebsachse entsprechend anzusteuern
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit USB Schnittstelle hergestellt ● RealSense-ROS Package installiert ● Workspace und ROS Installation gesourced ● ROS Package Abhängigkeit geometry_msgs, std_msgs, rospy
Testfalldaten	cmd_vel Daten: cmd_vel.linear.x = 1.0 cmd_vel.linear.y = 0.0 cmd_vel.angular.z = 1.0 cmd_vel.linear.x = 2.0 cmd_vel_linear.y = 0.0 cmd_vel.angular.z = -1
Erwartetes Verhalten	Es wird ein geschlossener Kreis gefahren. Zunächst vorwärts im Uhrzeigersinn, im Anschluss rückwärts gegen den Uhrzeigersinn

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.4 Fernsteuerung mit einem Gamepad

Testfall	**Beschreibung**
Testfall-Nummer	● 04
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Fernsteuerung (Teleoperation)
Testziel	● Der Jetracer lässt sich leicht und zuverlässig mit einem Gamepad installieren
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit Bluetooth oder 2,4GHz Empfänger ist hergestellt ● Joy ROS Package installiert ● Workspace und ROS Installation gesourced
Testfalldaten	● Joy-Message: joy.axes[1] = 0.5 joy.axes[4] = 1.0 joy. buttons[4] = 1
Erwartetes Verhalten	● Der Jetracer fährt einen geschlossenen Kreis

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.5 Autonome Navigation & Umgebungserkundung

Testfall	**Beschreibung**
Testfall-Nummer	● 05
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● ROS Navigation
Testziel	● Der Fennec kann selbstständig (ohne Fernsteuerung) Ziele im Raum erreichen. Außerdem kann eine Karte der Umgebung erzeugt werden die zur Pfadplanung dient. Der Roboter ist in der Lage sich auf dieser Karte zu lokalisieren.
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● ROS Navigation Package installiert ● AMCL installiert ● Gmapping installiert ● Workspace und ROS Installation gesourced ● Alle benötigten Sensoren für Gmapping fahren auf dem Roboter mit (mind. ein Laserscanner) ● Die Koordinatentransformationen aller Komponenten (Räder, Gelenke, Sensorik) sind vorhanden und werden gepublished ● Es werden Odometrie - Daten (Bewegungsdaten) vom Roboter gemessen (alternativ: kalkuliert, approximiert) und gepublished ● Der Roboter verfügt über einen Hardware Controller welche Twist-Messages entgegennehmen kann und in Bewegung umsetzt
Testfalldaten	●
Erwartetes Verhalten	● In Rviz wird nach und nach eine Karte erstellt und der Roboter plant einen Pfad zu jedem Ziel, welches vorgegeben wird

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.6 Ballerkennung

Testfall	**Beschreibung**
Testfall-Nummer	06
Testart	Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	Intel Realsense D415, OpenCV
Testziel	Die Intel Realsense kann über ROS gestartet werden. Bilder, die die Realsense permanent aufnimmt, können als Sensordaten an ROS übergeben werden. Diese werden über OpenCV verarbeitet, um im Bild einen Ball zu erkennen.
Testvoraussetzungen	● ROS Installation auf Ubuntu ● ROS Realsense package von Intel installiert ● Verbindung mit USB Schnittstelle hergestellt ● ROS cvbridge installiert
Testfalldaten	● Realsense über roslaunch realsense2_camera rs_camera.launch starten ● Ballerkennung über rosrun opencv find_ball.py starten ● Um ein Bild auf dem Bildschirm zu sehen, muss rosrun rqt_image_view rqt_image_view (2 mal in Folge ist richtig!) gestartet werden. Als „Topic“ muss /blob/image_blob ausgewählt werden.
Erwartetes Verhalten	● Erzeugen eines Live Feeds der Kamera. Wenn ein (blauer) Ball ins Bild gehalten wird, wird der Ball erkannt und ein roter Kreis wird um den Ball gezeichnet.

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwer[2]	
Bemerkung	Die Intel Realsense kann ohne Probleme gestartet werden. Der Ball lässt sich ohne Probleme erkennen.	
Tester Kunde	Tester Auftragnehmer	Datum
Umut Uzunoglu	Umut Uzunoglu	21.07.21

[2] Die Beschreibung der Fehlerkategorien entnehmen Sie bitte dem beigefügten Anhang

2.7 Ball verfolgen/anfahren

Testfall	Beschreibung
Testfall-Nummer	07
Testart	Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	Intel RealSense, OpenCV, Jetracer/Fennec
Testziel	Nach der Erkennung eines Balles, muss die Position des Balles im Bezug zur Kamera festgestellt werden und kontinuierlich aktualisiert werden. Entsprechend der Position des Balles im Bild (links oder rechts vom Zentrum des Bildes), wird dem Fennec eine Twist Message übergeben. Diese Twist Messages werden solange übergeben, wie der Bald auf dem Bild erkennbar ist. Auch soll es möglich sein, einen Ball zu erkennen der in Bewegung ist.
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit USB Schnittstelle hergestellt ● RealSense-ROS Package installiert ● Workspace und ROS Installation gesourced ● Testfall 2.1 muss funktionieren ● Umwandlung der x-y- Koordinaten vom Live-Bild in Twist Messages ● Übergabe der Twist Messages an Fennec
Testfalldaten	● Anschließen der Kamera an den Fennec ● Starten der Kamera mit “ roslaunch realsense2_camera rs_camera.launch” ● Starten der Ballerkennung über rosrun opencv find_ball.py ● Starten der Umwandlung von Koordinaten auf dem Bild in Twist Messages über rosrun opencv chase.py ● Blauer Ball als Erkennungsobjekt stillstehend ● Blauer Ball als Erkennungsobjekt in Bewegung
Erwartetes Verhalten	● Der Fennec fährt auf den Ball zu und korrigiert, falls die Kamera nicht mehr auf den Ball gerichtet ist. ● Rollt der Ball vom Fennec weg, verfolgt er ihn.

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwerwiegend	
Bemerkung	Da die Intel Realsense mehrere Kameras besitzt (Infrarot und Tiefen), muss die Position der Farbkamera mit beachtet werden. Die Farbkamera (RGB) ist bei der Intel Realsense D415 nicht mittig.	
Tester Kunde Umut Uzunoglu	Tester Auftragnehmer Umut Uzunoglu	Datum 21.07.21

2.8 Distanzberechnung relativ zur Kamera

Testfall	**Beschreibung**
Testfall-Nummer	● 08
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Intel Realsense
Testziel	● Die Distanz zwischen der Kamera und dem Objekt vor der Kamera berechnen und als Sensordaten an ROS weitergeben
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Verbindung mit USB Schnittstelle hergestellt ● Realsense-ROS Package installiert
Testfalldaten	Verschiedene Objekte/Wände auf verschiedenen Positionen und Entfernungen gegenüber der Tiefenkamera setzen
Erwartetes Verhalten	Ausgabe des Abstandes zum Objekt in Metern

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwerwiegend	
Bemerkung	Die Distanz zum Objekt wird vom Bildmittelpunkt aus bestimmt.	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21

2.9 Odometrie

Testfall	Beschreibung
Testfall-Nummer	● 09
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Jetracer, Lokalisierung
Testziel	● Durch Odometriedaten ist es möglich, die Lokalisierung des Fennecs auf der erstellten Karte zu verbessern.
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● Ansteuerung mit GamePad oder simples publizieren von /cmd_vel Werten, um den Fennec in Bewegung zu setzen muss möglich sein
Testfalldaten	● Twist-Message: cmd_vel.linear.x = 1.0 cmd_vel.linear.y = 0.0 cmd_vel.angular.z = 0 cmd_vel.linear.x = 0.5 cmd_vel.linear.y = 0.0 cmd_vel.angular.z = 0
Erwartetes Verhalten	● Für cmd_vel.linear.x wird 2,1m Distanz zurückgelegt

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.10 Inbetriebnahme ist einfach und schnell

Testfall	**Beschreibung**
Testfall-Nummer	● 10
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Jetracer, ROS Navigation, OpenCV
Testziel	● Startzeiten ermitteln
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● ROS Nodes vorbereitet und bereit zum starten
Testfalldaten	● Starten des Fennecs: o 20 Sekunden ● ROS Navigation: o Realsense starten/LiDAR starten (5 Sekunden) o Bei Bedarf GUI über Laptop starten (ca. 5 Sekunden) o AMCL starten (5 Sekunden) o Gmapping starten (ersten Scan registrieren ca 5 Sekunden) o Navigation stack (Move_base) starten (10 Sekunden) o Racecar.py auf dem Jetson Nano starten (ca. 10 Sekunden) **o Ca. 30-40 + Fennec starten** ● Ballerkennung/-verfolgung: o Realsense starten/LiDAR starten (5 Sekunden) o Find_ball.py starten (2 Sekunden) o Chase.py starten (2 Sekunden) o Racecar.py starten (ca. 10 Sekunden) o **Ca. 20 Sekunden**
Erwartetes Verhalten	● Der Racer braucht im Schnitt nicht länger als 2 Minuten zum Starten

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwerwiegend	
Bemerkung	Das Vorbereiten von Terminals über SSH ist nicht mit einberechnet, da das nach einmaligem Setup nicht mehr viel Zeit in Anspruch nimmt (im Terminal eine Kommandozeile + Passwort sind keine 3 Sekunden)	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21

2.11 Tore erkennen und auseinanderhalten

Testfall	**Beschreibung**
Testfall-Nummer	● 11
Testart	● Systemtest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Jetracer, OpenCV
Testziel	● Ein vorab definiertes Tor wird erkannt und lokalisiert
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● ROS Nodes vorbereitet und bereit zum starten ● Bild mit Feature Punkten des Tores vorab aufgenommen ● Bild- und TF-Framename sind identisch ● Autonomer Erkundungsmodus wird verwendet ● Stabile und schnelle WLAN Verbindung von Laptop zu Jetson ● SLAM Gmapping läuft
Testfalldaten	● Im autonomen Modus wird die Umgebung erkundet ● Das Tor ist an einer für den Roboter erreichbaren Stelle platziert. ● Sollte es zwei Tore geben, sollten diese sich in den Feature Punkten unterscheiden.
Erwartetes Verhalten	● Nachdem das Tor im Bild Frame ist und in 1,5 bis 2 m Distanz ist sollte in der GUI ein blauer Marker erscheinen der die Position und Orientierung des Tores darstellt.

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.12 Wartbarkeit

Testfall	**Beschreibung**
Testfall-Nummer	● 12
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Chassis
Testziel	● Funktionalität sicherstellen
Testvoraussetzungen	● Chassis ist gedruckt und montiert
Testfalldaten	● Man kann ohne Probleme das Ladekabel an die Ladebuchse des Waveshare Expansion Boards anschließen
Erwartetes Verhalten	● Der Jetracer lädt. Sichtbar an der rot aufleuchtenden LED am Ladegerät.

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwerwiegend	
Bemerkung		
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 30.06.21

2.13 Hindernisse umfahren

Testfall	**Beschreibung**
Testfall-Nummer	● 13
Testart	● Systemtest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Jetracer, ROS Navigation, LiDAR
Testziel	● Testen der Vermeidung von statischen Hindernissen
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● ROS Nodes vorbereitet und bereit zum starten ● LiDAR montiert ● ROS Navigation Stack läuft ● SLAM Gmapping läuft
Testfalldaten	● Erzeugen der Umgebungskarte ● Setzen eines Navigationszieles in freie Fläche ● Der Weg zum Ziel ist jedoch komplex und mit mehreren Hindernissen im Weg
Erwartetes Verhalten	● Der Jetracer findet einen Plan und fährt diesen Plan ab. Während der Fahrt wird die Abweichung von geplanten Pfad nachgeregelt und nachjustiert.

Testergebnis	X Bestanden ☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht ☐ Mittel ☐ Schwerwiegend	
Bemerkung	Es werden niemals Pfade geplant und ausgeführt die zur Kollision führen. Sollte es während dem Abfahren des Pfades zur Abweichungen/Problemen kommen stoppt der Roboter bevor er zusammen stößt.	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21

2.14 Gegen Ball fahren und schießen

Testfall	**Beschreibung**
Testfall-Nummer	• 14
Testart	• Systemtest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	• Jetracer, OpenCV
Testziel	• Sobald ein Ball erkannt wird, steuert der Fennec auf den Ball zu und schießt ihn.
Testvoraussetzungen	• Vollständige ROS Installation auf Ubuntu • Catkin Workspace erstellt • ROS Nodes vorbereitet und bereit zum starten • Intel RealSense gestartet • OpenCV chase_ball.py gestartet • Der Ball wurde im Vorfeld definiert und die Parameter für Form und Farbe sind für die Erkennung optimiert worden.
Testfalldaten	• Blauer Ball wird ins den Kamera Frame gehalten.
Erwartetes Verhalten	• Je nachdem wo sich der Ball im Bild befindet wird nach rechts oder links gesteuert, um den Ball möglichst zentral zu treffen.

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

2.15 Tor schießen

Testfall	Beschreibung
Testfall-Nummer	• 15
Testart	• Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	• Jetracer, ROS Navigation, LiDAR, SLAM Gmapping, Intel RealSense, Goal Scoring, OpenCV Bilderkennung
Testziel	• Der Spielball wird ins richtige Tor befördert
Testvoraussetzungen	• Vollständige ROS Installation auf Ubuntu • Catkin Workspace erstellt • ROS Nodes vorbereitet und bereit zum starten • Testfälle 5-8, 11, 13 & 14 sind bestanden
Testfalldaten	• Fennec wird im Spielfeld gestartet. Tor und Ball sind an erreichbaren Positionen platziert.
Erwartetes Verhalten	• Nachdem Ball und Tor erkannt wurden, navigiert der Fennec hinter den Ball in Richtung Tor und schießt ihn gerade in Richtung des Tores. Danach erfolgt ein Replanning mit einer kurzen Verzögerung, um den Ball die Chance zu geben zur Ruhe zu kommen. Sollte der Ball im Tor sein, kehrt der Roboter an den Startpunkt zurück und wartet bis der Ball an eine andere Stelle gebracht wird.

Testergebnis	☐Bestanden X Nicht Bestanden	
Fehlerkategorie	☐ Leicht X Mittel ☐ Schwerwiegend	
Bemerkung	Mangelnder Platz ermöglichen keine ausreichende Wendemöglichkeit für den Roboter um gut Pfade zwischen Tor und Ball planen zu können und sich richtig ausrichten zu können.	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21

2.16 Sensorfusion

Testfall	**Beschreibung**
Testfall-Nummer	● 16
Testart	● Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	● Intel Realsense, LiDAR, Odometrie
Testziel	● Verbesserung der Lokalisierung
Testvoraussetzungen	● Vollständige ROS Installation auf Ubuntu ● Catkin Workspace erstellt ● ROS Nodes vorbereitet und bereit zum starten# ●
Testfalldaten	● IMU der Realsense, Laserscan, und errechnete Odometrie werden von AMCL benutzt, um die Pose besser schätzen zu können
Erwartetes Verhalten	● Die Lokalisierung in der SLAM Karte funktioniert genauer und schneller.

Testergebnis	X Bestanden	☐ Nicht Bestanden	
Fehlerkategorie	☐ Leicht	☐ Mittel	☐ Schwerwiegend
Bemerkung			
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 21.07.21	

3 Testprotokoll

Testfall Nr.	Datum	Status	Schweregrad	Datum 2. Lauf	Status 2. Lauf
01	21.07.21	bestanden			
02	21.07.21	bestanden			
03	21.07.21	bestanden			
04	21.07.21	bestanden			
05	21.07.21	bestanden			
06	21.07.21	bestanden			
07	21.07.21	bestanden			
08	21.07.21	bestanden			
09	21.07.21	bestanden			
10	21.07.21	bestanden			
11	21.07.21	bestanden			
12	21.07.21	bestanden			
13	21.07.21	bestanden			
14	21.07.21	bestanden			
15	21.07.21	nicht bestanden	mittel	24.07.21	
16	21.07.21	bestanden			

4 Anhang

4.1 Fehlerkategorien

Für die Abnahme des Systems sind folgende Fehlerklassen definiert:

- **3 = Schwerer Mangel** — Produktivsetzung nicht möglich (Nachhaltige Störung des Softwareablaufes mit daraus resultierender Funktionsuntüchtigkeit des Systems bzw. Störung von Systemteilen, die zur Störung aller Arbeitsabläufe beim Auftraggeber führt.)
- **2 = Mittlerer Mangel** — Produktivsetzung möglich aber mangelhafte Funktionen nicht nutzbar (Durch eine Störung treten in Teilen der Programmabläufe nicht unerhebliche Störungen auf, so dass Teile der Software nicht verwendbar sind.)
- **1 = Leichter Mangel** — Produktivsetzung durch Workaround mit vertretbarem Zusatzaufwand möglich (Alle anderen als die in den vorstehenden Prioritätsgraden beschriebenen Störungsbilder)
-

1.1 Q-Kriterien ISO 9126

Gruppe	Q-Kriterium	
Funktionalität		
Sind alle im Pflichtenheft aufgeführten Kriterien vorhanden und ausführbar?	**Angemessenheit**	Merkmale von Software, die sich auf das Vorhandensein und die Eignung einer Menge von Funktionen für spezifizierte Aufgaben beziehen.
	Richtigkeit	Merkmale von Software, die sich beziehen auf das Liefern der richtigen oder vereinbarten Ergebnisse oder Wirkungen.
	Interoperabilität	Merkmale von Software, die sich auf ihre Eignung beziehen, mit vorgegebenen Systeme zusammenzuwirken.
	Ordnungsmäßigkeit	Merkmale von Software, die bewirken, dass die Software anwendungsspezifische Normen oder Vereinbarungen oder gesetzliche Bestimmungen oder ähnliche Vorschriften erfüllt.
	Sicherheit	Merkmale von Software, die sich auf ihre Eignung beziehen, unberechtigten Zugriff, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern.

Zuverlässigkeit		
Zu welchem Grad erfüllt die Software dauerhaft und korrekt die geforderten Funktionen?	**Reife**	Merkmale von Software, die sich auf die Häufigkeit von Versagen durch Fehlzustände in der Software beziehen.
	Fehlertoleranz	Merkmale von Software, die sich auf ihre Eignung beziehen, ein spezifiziertes Leistungsniveau bei Software-Fehlern oder Nicht-Einhaltung ihrer spezifizierten Schnittstelle zu bewahren.
	Wiederherstellbarkeit	Merkmale von Software, die sich beziehen auf die Möglichkeit, bei einem Versagen ihr Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen, und auf die dafür benötigte Zeit und den benötigten Aufwand.
Benutzbarkeit		
Wie schnell kann man den Umgang mit der Software lernen und wie leicht ist sie zu bedienen?	**Verständlichkeit**	Merkmale von Software, die sich auf den Aufwand für den Benutzer beziehen , das Konzept und die Anwendung zu verstehen.
	Erlernbarkeit	Merkmale von Software, die sich auf den Aufwand für den Benutzer beziehen, ihre Anwendung zu erlernen. (z.B. Ablaufsteuerung, Eingabe, Ausgabe)
	Bedienbarkeit	Merkmale von Software, die sich auf den Aufwand für den Benutzer bei der Bedienung und Ablaufsteuerung beziehen.
Effizienz		
Wie sind zeitliches Verhalten und Ressourcenverbrauch bei gegebenen System-voraussetzungen?	**Zeitverhalten**	Merkmale von Software, die sich beziehen auf die Antwort- und Verarbeitungszeiten und auf den Durchsatz bei der Ausführung ihrer Funktionen.
	Verbrauchsverhalten	Merkmale von Software, die sich darauf beziehen, wie viele Betriebsmittel bei der Erfüllung ihrer Funktionen benötigt werden und wie lange.
Änderbarkeit		
Mit welchem Zeit- und Arbeitsaufwand lassen sich Änderungen sowie Fehlererkennung und -behebung durchführen?	**Analysierbarkeit**	Merkmale von Software, die sich auf den Aufwand beziehen, der notwendig ist, um Mängel oder Ursachen von Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen.
	Modifizierbarkeit	Merkmale von Software, die sich auf den Aufwand beziehen, der zur Ausführung von Verbesserungen, zur Fehlerbeseitigung oder zur Anpassung an Umgebungsänderungen notwendig ist.
	Stabilität	Merkmale von Software, die sich auf das Risiko unerwarteter Wirkungen von Änderungen beziehen.
	Prüfbarkeit	Merkmale von Software, die sich auf den Aufwand beziehen, der zur Prüfung der geänderten Software notwendig ist.
Übertragbarkeit		
Mit welchem Aufwand lässt sich die Software an geänderte/ verbesserte Systembedingungen anpassen bzw. in neuen Systemen einsetzen?	**Anpassbarkeit**	Merkmale von Software, die sich auf die Möglichkeit beziehen, sie an verschiedene festgelegte Umgebungen anzupassen, wenn nur Schritte unternommen oder Mittel eingesetzt werden, die für diesen Zweck für die betrachtete Software vorgesehen sind.
	Installierbarkeit	Merkmale von Software, die sich auf den Aufwand beziehen, der zur Installation der Software in einer festgelegten Umgebung notwendig ist.
	Konformität	Merkmale von Software, die bewirken, dass die Software Normen oder Vereinbarungen zur Übertragbarkeit erfüllt.

	Austauschbarkeit	Merkmale von Software, die sich beziehen auf die Möglichkeit, diese anstelle einer anderen Software in der Umgebung jener Software zu verwenden und auf den dafür notwendigen Aufwand.

1.2 Q-Kriterien für Dokumente

Für die Erreichung des Projektzieles, das Produkt „Dokument“ zu erzeugen, dass den fachlichen und technischen Anforderungen des Auftraggebers entspricht, ergeben sich z.B. die folgenden Qualitätsmerkmale:

Merkmal	Erläuterung	Mindest-anfordrg.	Prüfmöglichkeit
Eindeutigkeit	Eignung von Dokumenten zur unmissverständlichen Vermittlung von Informationen für jeden Leser		Keine offenen Fragen zu den einzelnen Abschnitten (Prüfung durch Gruppeninspektion und Diskussion)
Lesbarkeit	Eignung von Dokumenten zur Entnahme der darin enthaltenen Informationen		Prüfung durch Einsatz eines unbedarften Testlesers, Vorhandensein eines Glossars, Erläuterung von Fachbegriffen
Verständlichkeit	Eignung von Dokumenten zur erfolgreichen Vermittlung der darin enthaltenen Informationen an einen sachkundigen Leser		Vorhandensein eines Glossars, Integration von Illustrationen, Diagrammen
Detaillierungsgrad	Vorhandensein der ausreichenden Beschreibung der fachlichen und technischen Einzelheiten im Dokument		Beschreibung der Sonder- und Ausnahmefälle, gleiche Behandlung (gleiche Detaillierung) aller Textabschnitte
Funktionale Vollständigkeit	Vorhandensein der für den Zweck der Dokumentation notwendigen und hinreichenden Information		Einsatz des <KUNDE>Templates gewährleistet die Voll-ständigkeit an notwendigen Informationen, Beschreibung der Sonder- und Ausnahmefälle
Fehlerfreiheit	Nichtvorhandensein von sprachlichen Fehlern, die die Informationsaufnahme beeinträchtigen		Rechtschreib- und Grammatikprüfung
Widerspruchsfreiheit	Nichtvorhandensein von einander entgegenstehenden Aussagen im Dokument		Unnötige Redundanzen sollen vermieden werden, Dokument soll in sich konsistent sein
Aktualität	Übereinstimmung der Beschreibung der Situation in Dokument und Wirklichkeit		Gespräche mit dem Auftraggeber (Kundeninspektion, Workshops)
Funktionale Korrektheit	Nichtvorhandensein von funktionalen Fehlern, die den fachlichen und technischen Inhalt betreffen		Wiedergabe der Anforderungen aus dem Vorgängerdokument
Normenkonformität	Erfüllung der für die Erstellung von Dokumenten geltenden Vorschriften und Normen		Einsatz des <KUNDE>Templates gewährleistet die formale Richtigkeit
Änderbarkeit	Eignung von Dokumenten zur Ermittlung aller von einer Änderung betroffenen Dokumententeile und zur Durchführung der Änderung		Einsatz des <KUNDE>Templates gewährleistet die formale Änderbarkeit, unnötige Redundanzen sollen vermieden werden