

Qualitätssicherung

Fennec Racer

Autor: Team Fennec Bot
Letzte Änderung: 30.06.2021
Dateiname: Qualitätssicherung_Fennec_Bot.docx
Version: 1.0

Inhaltsverzeichnis

1 Testplan	4
2 Testfälle	7
Einbindung RPLIDAR mit ROS	7
Einbindung Intel Real Sense mit ROS	8
Verwendung eines Hardware Controllers für Ackerman Steuerung	9
Fernsteuerung mit einem Gamepad	10
Autonome Navigation & Umgebungserkundung	12
3 Testprotokoll	12
4 Anhang	13
Fehlerkategorien	13
Q-Kriterien ISO 9126	13
Q-Kriterien für Dokumente	15

Copyright

Team Fennec Bot

Die Weitergabe, Vervielfältigung oder anderweitige Nutzung dieses Dokumentes oder Teile davon ist unabhängig vom Zweck oder in welcher Form untersagt, es sei denn, die Rechteinhaber/In hat ihre ausdrückliche schriftliche Genehmigung erteilt.

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 Ballerkennung

Pro Testfall soll das folgende Template angewandt werden:

Testfall	Beschreibung
Testfall-Nummer	01
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	<ul style="list-style-type: none"> • ROS Installation auf Ubuntu • ROS Realsense package von Intel installiert • Verbindung mit USB Schnittstelle hergestellt • ROS cvbridge installiert
Testfalldaten	<ul style="list-style-type: none"> • Realsense über roslaunch realsense2_camera rs_camera.launch starten • Ballerkennung über rosruncv find_ball.py starten • Um ein Bild auf dem Bildschirm zu sehen, muss rosruncv rqt_image_view rqt_image_view (2 mal in Folge ist richtig!) gestartet werden. Als „Topic“ muss /blob/image_blob ausgewählt werden.
Erwartetes Verhalten	<ul style="list-style-type: none"> • Erzeugen eines Livefeeds der Kamera. Wenn ein (blauer) Ball ins Bild gehalten wird, wird der Ball erkannt und ein roter Kreis wird um den Ball gezeichnet.

Testergebnis	<input checked="" type="checkbox"/> Bestanden <input type="checkbox"/> Nicht Bestanden	
Fehlerkategorie	<input type="checkbox"/> Leicht <input type="checkbox"/> Mittel <input type="checkbox"/> Schwer ¹	
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	29.06.21

¹ Die Beschreibung der Fehlerkategorien entnehmen Sie bitte dem beigefügten Anhang

2.2 Ball verfolgen/anfahren

Testfall	Beschreibung
Testfall-Nummer	02
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 Ball auf dem Bild erkennbar ist. Auch soll es möglich sein, einen Ball zu erkennen der in Bewegung ist.
Testvoraussetzungen	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Anschließen der Kamera an den Fennec • Starten der Kamera mit "roslaunch realsense2_camera rs_camera.launch" • Starten der Ballerkennung über rosrund opencv find_ball.py • Starten der Umwandlung von Koordinaten auf dem Bild in Twist Messages über rosrund opencv chase.py • Blauer Ball als Erkennungsobjekt stillstehend • Blauer Ball als Erkennungsobjekt in Bewegung
Erwartetes Verhalten	<ul style="list-style-type: none"> • Der Fennec fährt auf den Ball zu und korrigiert, falls die Kamera nicht mehr auf den Ball gerichtet ist. • Rolllt der Ball vom Fennec weg, verfolgt er ihn.

Testergebnis	<input checked="" type="checkbox"/> Bestanden <input type="checkbox"/> Nicht Bestanden	
Fehlerkategorie	<input type="checkbox"/> Leicht <input type="checkbox"/> Mittel <input type="checkbox"/> 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 29.06.21

2.3 Distanzberechnung relativ zur Kamera

Testfall	Beschreibung
Testfall-Nummer	• 03
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	<ul style="list-style-type: none"> • 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	<input checked="" type="checkbox"/> Bestanden <input type="checkbox"/> Nicht Bestanden	
Fehlerkategorie	<input type="checkbox"/> Leicht <input type="checkbox"/> Mittel <input type="checkbox"/> Schwerwiegend	
Bemerkung	Die Distanz zum Objekt wird vom Bildmittelpunkt aus bestimmt.	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 30.06.21

2.4 Odometrie

Testfall	Beschreibung
Testfall-Nummer	<ul style="list-style-type: none"> 04
Testart	<ul style="list-style-type: none"> Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	<ul style="list-style-type: none"> Jettracer, Lokalisierung
Testziel	<ul style="list-style-type: none"> Durch Odometriedaten ist es möglich, die Lokalisierung des Fennecs auf der erstellten Karte zu verbessern.
Testvoraussetzungen	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Twist-Message: <ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Für cmd_vel.linear.x wird 2,1m Distanz zurückgelegt

Testergebnis	<input type="checkbox"/> Bestanden <input checked="" type="checkbox"/> Nicht Bestanden	
Fehlerkategorie	<input checked="" type="checkbox"/> Leicht <input type="checkbox"/> Mittel <input type="checkbox"/> Schwerwiegend	
Bemerkung	Es werden 2,3m anstatt der tatsächlich 2,1m langen Strecke als Odometriedaten publiziert. Abweichungen der Odometrie verschlechtern die Qualität der Belegungskarte und die Lokalisierung. Durch die Mitverarbeitung von Sensordaten kann ein Abdriften von der tatsächlichen Position in der Karte jedoch vermieden werden.	
Tester Kunde Lukas Evers	Tester Auftragnehmer Lukas Evers	Datum 30.06.21

2.5 Startup

Testfall	Beschreibung
Testfall-Nummer	<ul style="list-style-type: none"> • 04
Teststart	<ul style="list-style-type: none"> • Funktionstest
Zu testender Geschäftsprozess/ Zu testende Funktionsgruppe	<ul style="list-style-type: none"> • Jetracer, ROS Navigation, OpenCV
Testziel	<ul style="list-style-type: none"> • Startzeiten ermitteln
Testvoraussetzungen	<ul style="list-style-type: none"> • Vollständige ROS Installation auf Ubuntu • Catkin Workspace erstellt • ROS Nodes vorbereitet und bereit zum starten
Testfalldaten	<ul style="list-style-type: none"> • Starten des Fennecs: <ul style="list-style-type: none"> ○ 20 Sekunden • ROS Navigation: <ul style="list-style-type: none"> ○ Realsense starten/LiDAR starten (5 Sekunden) ○ Bei Bedarf GUI über Laptop starten (ca. 5 Sekunden) ○ AMCL starten (5 Sekunden) ○ Gmapping starten (ersten Scan registrieren ca 5 Sekunden) ○ Navigation stack (Move_base) starten (10 Sekunden) ○ Racecar.py auf dem Jetson Nano starten (ca. 10 Sekunden) ○ Ca. 30-40 + Fennec starten • Ballerkennung/-verfolgung: <ul style="list-style-type: none"> ○ Realsense starten/LiDAR starten (5 Sekunden) ○ Find_ball.py starten (2 Sekunden) ○ Chase.py starten (2 Sekunden) ○ Racecar.py starten (ca. 10 Sekunden) ○ Ca. 20 Sekunden
Erwartetes Verhalten	<ul style="list-style-type: none"> • Der Racer braucht im Schnitt nicht länger als 2 Minuten zum Starten

Testergebnis	<input checked="" type="checkbox"/> Bestanden <input type="checkbox"/> Nicht Bestanden	
Fehlerkategorie	<input type="checkbox"/> Leicht <input type="checkbox"/> Mittel <input type="checkbox"/> 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 30.06.21

3 Testprotokoll

TestfallNr.	Datum	Status	Schweregrad	Datum 2. Lauf	Status 2. Lauf
01	30.06.21	bestanden			
02	30.06.21	bestanden			
03	30.06.21	bestanden			
04	30.06.21	nicht bestanden	leicht		
05	30.06.21	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 Vollständigkeit 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