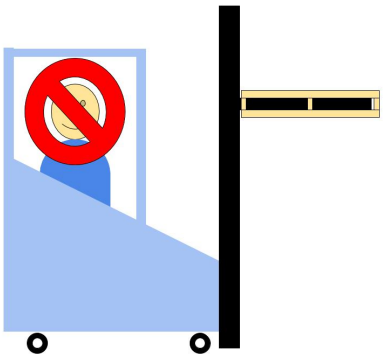
Användarhandledning
Autonom truck

Version 1.0

Redaktör: Joar Manhed
Datum: 18 december 2018



Status

Granskad	Lovisa Jansson	2018-12-13
Godkänd	Andreas Bergström	2018-12-14

Kursnamn:	Reglerteknisk projektkurs	E-post:	kimby803@student.liu.se
Projektgrupp:	TRUCK-HT18	Dokumentansvarig:	Joar Manhed
Kurskod:	TSRT10	Dokumentansvarigs e-post:	joama350@student.liu.se
Projekt:	Autonom truck	Dokumentnamn:	Användarhandledning.pdf

Projektidentitet

Grupp E-post:	kimby803@student.liu.se
Hemsida:	
Beställare:	Andreas Bergström, ISY, Linköping University **Telefon:** +46 10-711 54 54, **E-post:** andreas.bergstrom@liu.se
Kund:	Magnus Persson, Toyota Material Handling, Mjölby **Telefon:** +46 771-220 220 , **E-post:** magnus.persson@toyota-industries.eu
Kursansvarig:	Daniel Axehill, ISY, Linköping University **Telefon:** +46 13-28 40 42, **E-post:** daniel.axehill@liu.se
Projektledare:	Kim Byström
Handledare:	Erik Hedberg, ISY, Linköping University **Telefon:** +46 13-28 13 38, **E-post:** erik.hedberg@liu.se

Gruppmedlemmar

Namn	**Ansvarsområde**	**Telefon**	**E-post (@student.liu.se)**
Kim Byström	Projektledare	072-7432190	kimby803
Lovisa Jansson	Designansvarig	076-3906525	lovja529
Anton Johansson	Komponentansvarig rörelseplanering	076-1962818	antjo244
Joar Manhed	Dokumentansvarig	076-5865400	joama350
David Sandmark	Komponentansvarig modellering & simulering	073-7613213	davsa696
Niklas Stenberg	Komponentansvarig reglering	076-8018632	nikst888
Pär Sörliden	Mjukvaruansvarig	076-5955950	parso619
Gustaf Westerholm	Testansvarig	070-8257421	guswe541

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2018-12-07	Första utkast.	Alla	Kim Byström
0.2	2018-12-10	Ändringar efter kommentarer från handledare.	Pär Sörliden	Lovisa Jansson
0.3	2018-12-13	Ändringar efter kommentarer från beställare.	Lovisa Jansson	Lovisa Jansson
1.0	2018-12-14	Godkänd av beställare	-	-

Innehåll

1 Översikt

Den här handledningen är till för att guida användare av den simuleringsmiljö som tagits fram i projektet Autonom Truck. Simuleringen är gjord i spelmotorn Unity där den simulerade trucken visas i 3D. Utöver simuleringsmiljön är även två moduler för reglering och planering utvecklade. Dessa hämtar information från simuleringsmiljön och skickar sedan styrsignaler till simuleringen.

1.1 Utrustning

Systemet består av två stycken datorer av märket Raspberry Pi, en för navigering och en för reglering, samt en persondator. De Raspberry Pi som används som navigeringsklient respektive regleringsklient är märkt med en klisterlapp ”Toyota RP N1” samt ”Toyota RP C1”. På persondatorn körs Unity-simuleringn, vilken även agerar server vid kommunikationen. För att klienterna och servern ska kunna kommunicera behöver de vara uppkopplade till samma lokala nätverk (LAN). Detta görs förslagsvis med en router eller via nätverksdelning. För att data ska skickas korrekt inom nätverket krävs att serverns IP-adress skrivs in i TCPServer.cs på servern och i client.cpp hos klienten.

2 Simulering

2.1 Köra simulering

Systemet körs i följande steg:

1. Koppla upp datorn med Unity och båda Raspberry Pi (klienterna) på samma nätverk. (Eduroam fungerar inte för detta ändamål)
2. Starta Unity och öppna Unity-projektet i mappen *toyota/Simulering*.
3. Leta upp scriptet ”TCPServer.cs” i mappen *Communication*.
4. Leta upp filen ”client.cpp” på klienterna i mappen *toyota/Kommunicering*.
5. Kontrollera IP-adressen på datorn som kör Unity och mata in denna i ”client.cpp” och i ”TCPServer.cs”.
6. Kompilera navigeringsklienten genom att stå i mappen *toyota/Planering* och skriva *make* i en terminal. Filen *run.o* skapas.
7. Kompilera regleringsklienten på liknade sätt. Stå i mappen *toyota/Reglering* och skriv *make* i en terminal. Filen *run.o* skapas. Observera att detta steg för närvarande inte är nödvändigt då regleringen inte fungerar från klienten utan körs direkt i Unity via filerna i mappen *toyota/Simulering/Assets/Control*. Se 2.2 för att ändra så att regleringen körs via klienten.
8. Säkerställ att navigeringen har tillgång till en .txt-fil med uppdrag i form av x- och y-koordinater på de önskade destinationerna. Varje destination ska skrivas på en egen rad på formatet ”x,y;” där x är destinationens x-koordinat och y är destinationens y-koordinat. Döp filen till *uppdrag.txt* och lägg den i mappen *toyota/Planering*. Observera att i Unity kör trucken i xz-planet medan planering sker enligt smartness i xy-planet. En mappning sker mellan y och z.
9. Starta simuleringen i Unity, vilket även startar servern, genom att klicka på Play-knappen.
10. Starta planeringsklienten genom att skriva *./run.o* i mappen *toyota/Planering*.
11. Starta regleringsklienten genom att skriva *./run.o* i mappen *toyota/Reglering*. Observera att detta steg förnuvarande inte är nödvändigt då regleringen inte fungerar från klienten utan körs direkt i Unity via filerna i mappen *toyota/Simulering/Assets/Control*. Se 2.2 för att ändra så att regleringen körs via klienten.
12. I simuleringen kan du nu se den bana som trucken ska följa i svart och hur trucken faktiskt kör i vitt. Observera att den planerade banan som visas är den som fås direkt från planering och är ganska hackig och det är inte exakt den som trucken reglerar efter utan regulatorn mjukar till trajektorian internt och reglerar efter den tillmjukade trajektorian, vilken inte visas i Unity.

2.2 Byte till reglering via klient istället för direkt från Unity

För att köra regleringen via klienten i *toyota/Reglering* istället för via Unity behöver två filer ändras: *Connection.cs* i mappen *toyota/Simulering/Assets/Communication/* och *TruckModel.cs* i mappen *toyota/Simulering/Assets/Model/* genom att kod markerad med kommentaren *// Add this when porting to C++* läggs till samt att kod med kommentaren *// Remove when porting to C++* tas bort. Vidare behövs justeringar i C++ koden under *toyota/Reglering* för att få det att fungera (se mer i tekniska rapporten om det).

2.3 Val av regulator

Då regleringen körs via Unity väljs regulatorn i filen *TruckModel.cs* i mappen *toyota/Simulering/Assets/Model/* i funktionen *Start()*. Observera att då trajektorian fås från planeringsmodulen så fungerar för närvarande inte regulatorn *FeedForwardController* utan den har endast fåtts att fungera då trajektorian är fördefinierad. Regulatorn *AnglePIDController* rekommenderas och är den som fåtts att fungera med planering. Regulatorparameterar kan ändras i motsvarande filer i mappen *toyota/Simulering/Assets/Control/* om så önskas.

När regleringen körs från klienten i *toyota/Reglering* kan regulator väljas i filen *reglering.cpp* i mappen *toyota/Reglering*. Regulatorparameterar kan ändras i filerna *controller.h* och *controller.cpp* i mappen *toyota/Reglering* om så önskas.

2.4 Byte av modell

I simuleringen finns två olika modeller att välja mellan, *TruckSteerableCADModel* och *TruckIdealizedModel*. I Unity, i fönstret *Hierarchy*, under *SampleScene*, finns de två modellerna. Om du klickar på en av modellerna och tittar i *Inspector*-fönstret kan du se om modellen används eller inte. Om det finns en bock i rutan längst upp till vänster i detta fönster så används modellen annars används den inte. För att byta modell, bocka ur rutan och klicka sedan på modellnamnet du ska byta till i *Hierarchy*-fönstret, bocka sedan i rutan i *Inspector*-fönstret för denna modell.

2.5 Införande av brus

Om *TruckSteerableCADModel* är vald i *Hierarchy*-fönstret så kan de olika brusen sättas i *Inspector*-fönstret. Brusen består av vitt brus och det är dess standardavvikelse som kan sättas i *Inspector*-fönstret. Först väljs brus på mätsignalerna eller styrsignalerna. Detta görs genom att bocka i *Use_measuerment_noise_* eller *Use_control_noise_* i *Wheel Collider*-blocket i *Inspector*-fönstret. Sedan väljs standradavvikelse för de olika brusen *Position_std_dev_* (position), *Vs_std_dev_* (drivhjulshastighet) och *Alpha_std_dev_* (styrvinkel).

2.6 Dataloggning till Matlab

Vid slutet av varje körning sparas data till en csv-fil som sedan kan öppnas med ett Matlab-script. Scriptet heter ”readcsv.m”. Kör du detta script får du fram grafer över truckens faktiska position, hastighet, acceleration och ryck mot motsvarande önskade värden. Tillgängliga finns även v_s, α, w, u_1, u_2, ν_1, μ_1, μ_2. Se teknisk rapport för närmare beskrivning av beteckningarna. Denna dataloggning fungerar då regleringen körs via Unity och har inte testats för reglering via klient, men funktionaliteten där är påbörjad.

3 Programmera på klienterna

Det finns två sätt att göra ändringar på en klient:

1. Koppla Raspberry Pi:n till en datorskärm via HDMI och ett tangentbord via USB (alt. mus också) som i figur 1. På så sätt kan ändringar göras på samma sätt som på en vanlig persondator.

2. SSH-kommunikation mellan Raspberry Pi (RP) och persondator via ett lokalt nätverk. Först måste SSH-kommunikation vara tillåtet hos RP:n. I standardutförande är detta avslaget, men de två RP som använts i detta projekt efterlämnas med detta påslaget. För en RP med SSH avslaget behövs en skärm och tangentbord för att i terminalen skriva: *systemctl enable ssh* följt av *sudo systemctl start ssh*. Lämpliga program för SSH-kommunikation från Windows är PuTTY samt WinSCP. PuTTY fungerar som en teminal-emulator som tillåter bl.a. att kompilera och exekvera filer på RP:n. WinSCP ger tillgång till filstrukturen på RP:n och låter dig flytta över och redigera i filer. Detta använda således för att skriva kod.

Figur 1: Uppsättning av utrustning