Kapittel 1

Dokumentasjon: Navsys

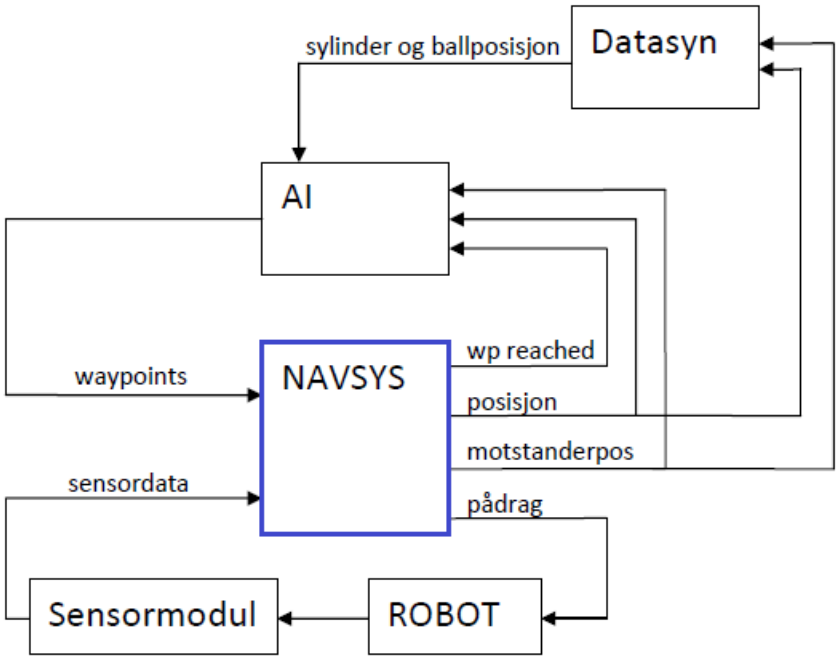
Navsys tar seg av navigering på spillebrettet. Det er her pådrag til motorene blir satt, og posisjonen estimeres. Navsys er i år helt redesignet.

1.1 Meldinger

Navsys kommuniserer med både AI, datasyn og gateway. Meldinger som blir sendt kan sees i figur 1.1. Pådrag blir sendt som pwm-signal gjennom gateway til motorene. Sensordata, i form av odometrimålinger, mottas fra sensormodulen gjennom gateway. Odometrimålingene oppdateres ca 100 ganger i sekundet. Navsys sender posisjon og motstanderposisjon til datasyn. Dette blir sendt med en gang det er beregnet da det er viktig for datasyn å motta posisjon så nøyaktig som mulig. Utover dette er det ingen kommunikasjon direkte mellom navsys og datasyn. Navsys og AI sender flere meldinger med hverandre. Posisjon og motstanderposisjon sendes fra navsys til AI ved hver iterasjon av while-løkken i main-metoden. Når roboten har navigert seg innenfor *circle-of-acceptance* til ønsket waypoint sender navsys *waypointReached* til AI. Når AI har mottatt *waypointReached*, eller når det har vært nødvendig å oppdatere strategien, vil det sendes nye waypoints til navsys. Waypoint-listen som sendes består av to waypoints. Det er alltid element 0 i denne listen som er ønsket waypoint.

1.2 Oppbygning

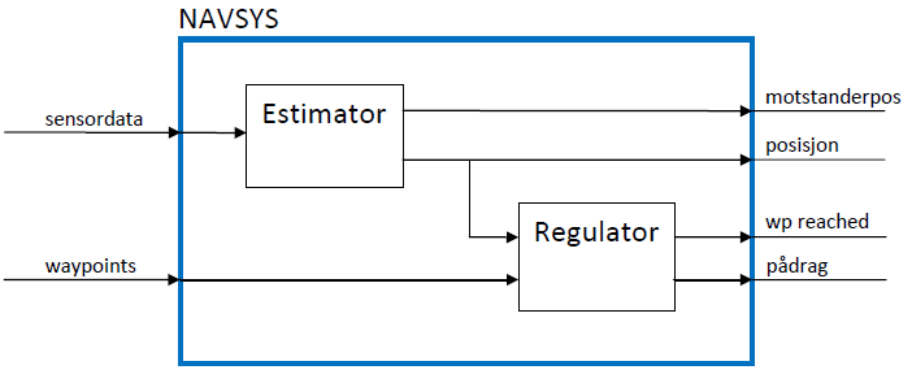
En oversikt over hvordan systemet er oppbygget kan sees i figur 1.2. Estimatoren behandler sensordata og estimerer posisjon og motstanderposisjon, mens regulatoren minimerer avstandsfeilen mellom ønsket posisjon og estimert posisjon.



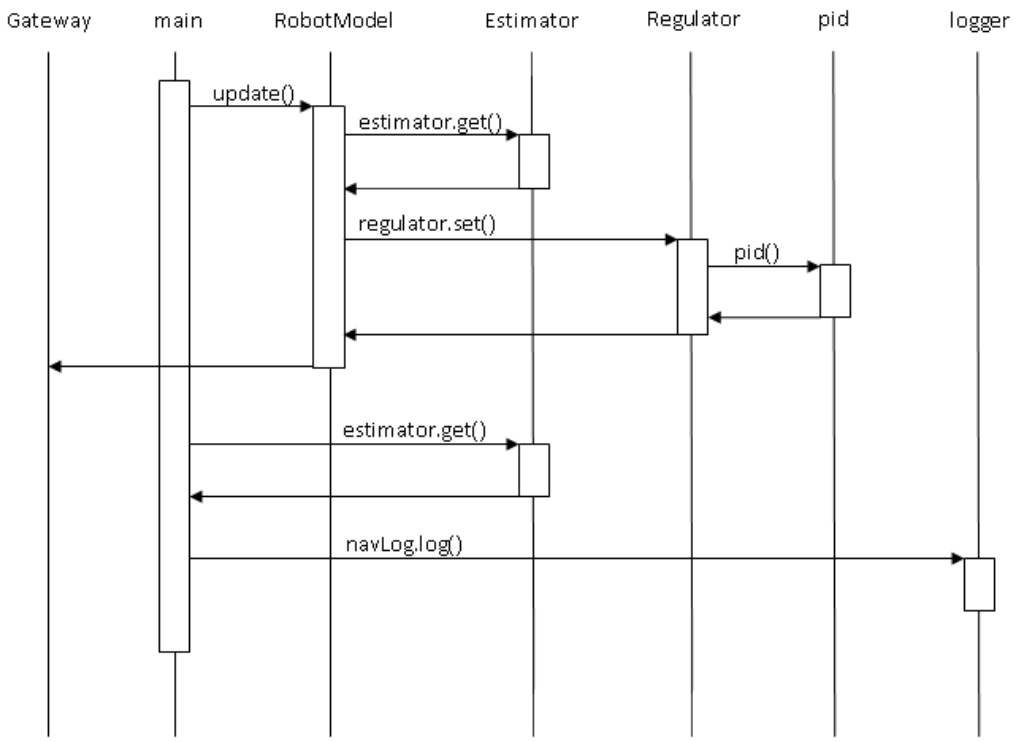
Figur 1.1: Kommunikasjon mellom modulene

Sekvensdiagram for regulatorsløyfen kan sees i figur 1.3. I main-metoden, som ligger i main.cpp, kjøres det initialiseringsmetoder før den setter igang RobotModel. RobotModel henter posisjonsestimat fra Estimator og kjører regulatoren. Dette skjer omtrent 100 ganger i sekundet. Main-metoden starter også logger, som er en klasse som tilhører gateway som logger ønskede data. Loggene har blitt brukt til å lage plot i matlab. Dette har vært svært nyttig for debugging.

Estimatoren kjøres fra kommunikasjonstråden da det er viktig at denne blir oppdatert etter hver gang gateway sender nye målinger.



Figur 1.2: Struktur i navsys-modulen



Figur 1.3: Sekvensdiagram

1.3 Oversikt over kildekode

main.cpp/.h

Main-metoden kjører initialiseringsmetoder og venter til det er mottatt startposisjon fra AI. Den kjører en while-løkke som kjører updateModel() i RobotModel og setter igang logger. Posisjon og motstanderposisjon sendes til AI herfra.

communication.cpp/.h

Communication.cpp kjører en egen tråd som mottar alle meldinger. De mottatte meldingene settes inn i ønskede objekter. Set()-metoden til Estimator kjører mår det er mottatt odometriinformasjon. Da sendes også estimert posisjon til datasyn. Motstanderposisjon sendes til datasyn etter at det er oppdatert. Alle metodene som brukes i programmet er opprettet i denne filen.

RobotModel.cpp/.h

I RobotModel ligger metoden updateModel() som er metoden som kjører Regulator og som sender pådrag til motorkortet. Pådraget fra Regulator er i form av translasjon og rotasjon. Dette blir omregnet til pådrag på venstre og høyre motor. Begrensninger på pådraget i tillegg til å øke pådraget hvis friksjonen er for høy skjer her før det sendes til motorkort.

Estimator.cpp/.h

Set()-metoden i Estimator-klassen kjøres hver gang odometridata blir oppdatert. Den tar inn odometridata som en struct og regner om dette til antall step kjørt siden forrige måling. Dette blir brukt til å berenge robotens rotasjon og lengde kjørt siden forrige måling. Robotens estimerte posisjon blir beregnet fra robotens posisjon ved forrige tidsskritt og forandring fra odometrimålingene.

Regulator.cpp/.h

Denne klassen holder oversikt over neste waypoint og pådrag. Set()-metoden kjører først next_waypoint() som regner om neste waypoint i *local*-koordinater til *body*-koordinater så avstand og vinkel til neste waypoint kan brukes videre. Så sjekker set()-metoden om ønsket waypoint er nådd. Den tar hensyn til om waypointet er retningsbestemt, om det skal kjøres i sirkelbane eller om det skal rygges til. Hvis waypointet er nådd, sendes

waypointReached() til AI. Hvis waypointet ikke er nådd kjøres pid(). Pid() er en metode i Regulator som oppretter to pid-objekter og kjører run()-metoden i pid.cpp på disse.

pid.cpp/.h

Pid er en klasse som består av en enkel regulator med proporsjonal, integral og derivatvirkning. Den har også noen begrensninger på output som kan settes i pid()-metoden i Regulator.

posix_messages.h

Inneholder *defines* til bruk i meldinger.

navsys_structs.h

Inneholder structer som brukes i navsys.

communication_structs.h

Inneholder strukter som blir brukt i kommunikasjon mellom systemer.

messageQueue.cpp/.h

Inneholder funksjoner for mottak av posix-meldinger.