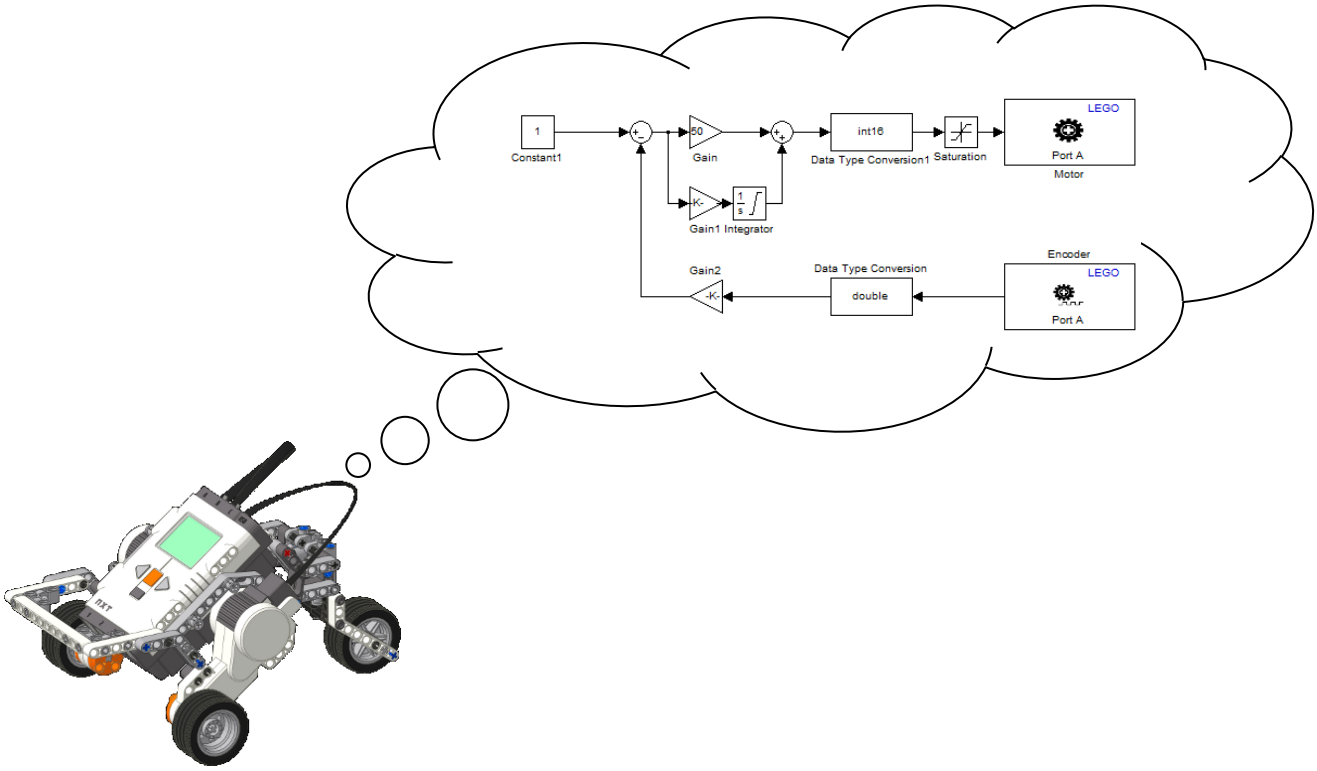
Teknostart

Introduksjon til MATLAB/Simulink

og LEGO Mindstorms



NTNU

Institutt for teknisk kybernetikk

August 2019

Versjon 2.4

INNHOLD

HVA ER DETTE?

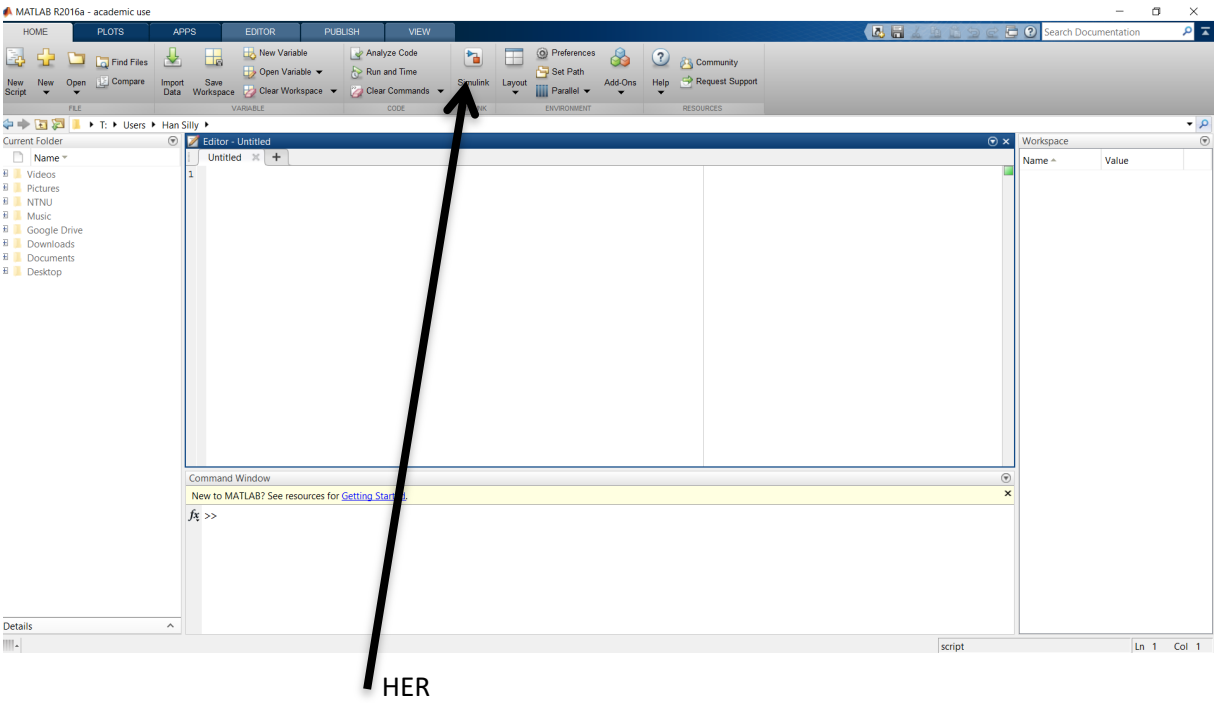
Her følger noen eksempler og frivillige oppgaver du kan bryne deg på hvis har lyst til å bli litt mer kjent med MATLAB/Simulink og LEGO Mindstorms. Du kan også lære noen triks som vil gjøre det lettere å komme seg gjennom banen.

Disse eksemplene ble opprinnelig laget for en tidligere versjon av Lego Mindstorms, kalt NXT. Derfor vil noen av skjermdumpene se litt annerledes ut enn for den nåværende Legoen. Simulinkblokkene som brukes, og måten å gjøre ting på for øvrig, er imidlertid helt lik. Om noe likevel ikke skulle gi mening, spør læringsassistenten din.

Lykke til!

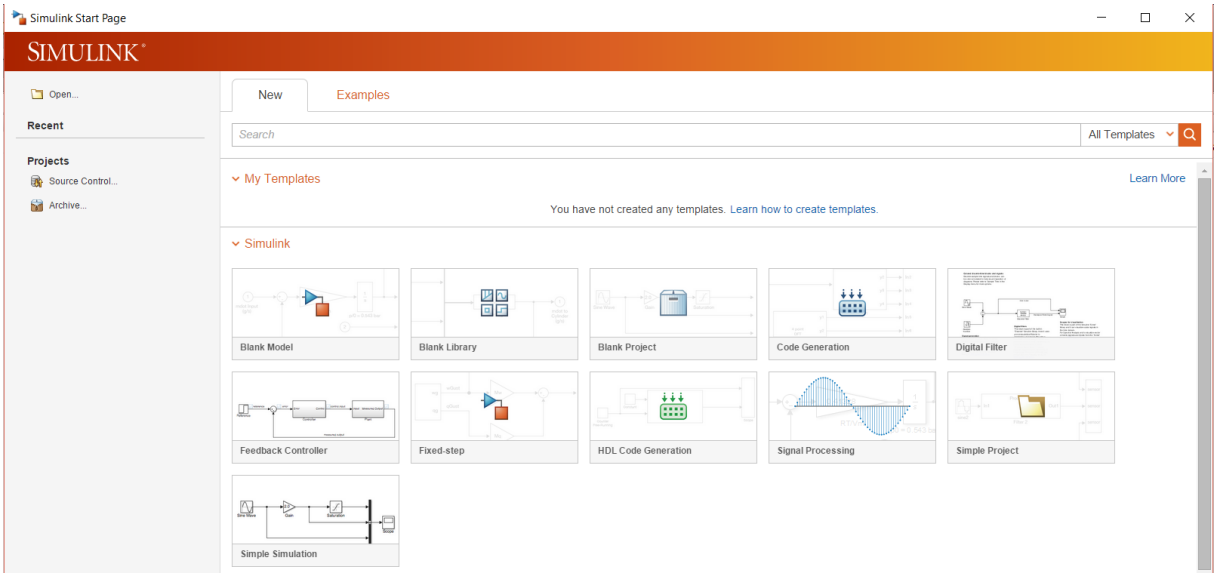
INTRODUKSJON TIL SIMULINK – STYRE EN MOTOR

Start Simulink. Det gjøres ved å trykke på denne  knappen i MATLAB-vinduet. Den befinner seg

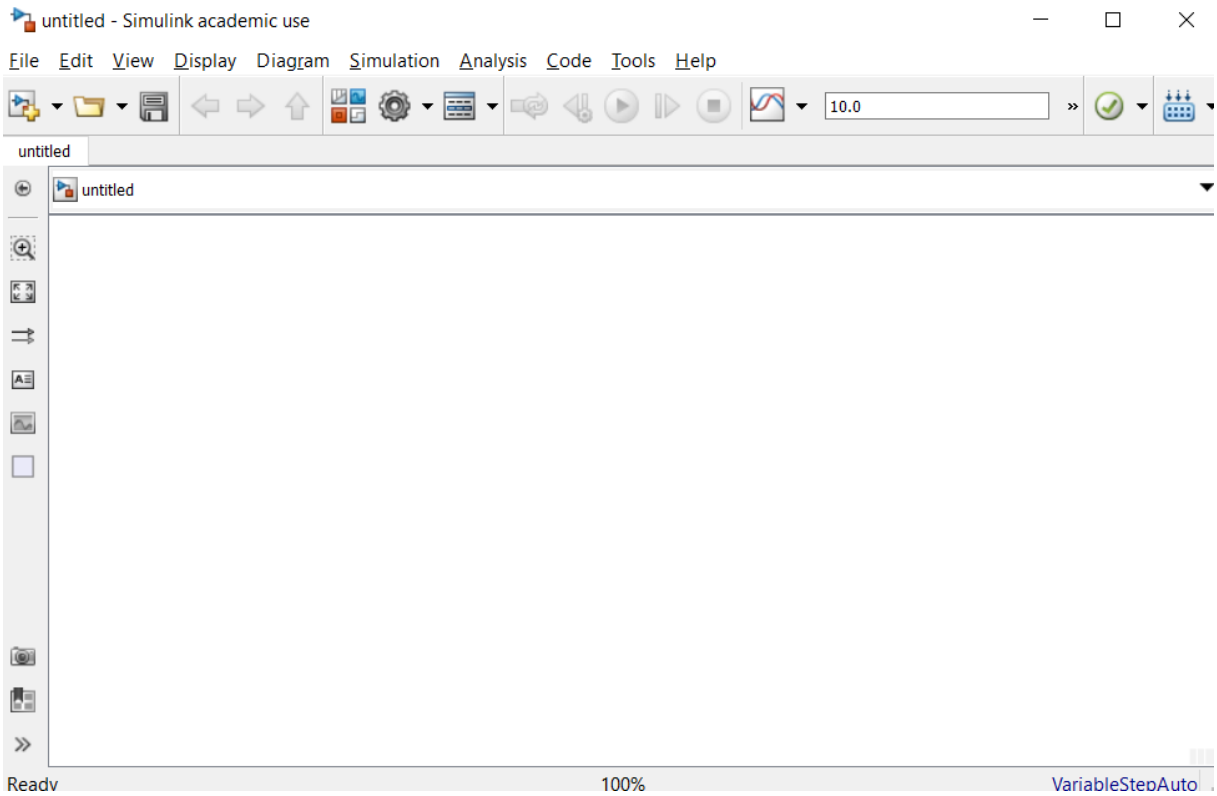


HER

Da vil Simulink sin startside kommer opp. Start en ny modell enten ved å trykke «ctrl+N», eller ved å velge «Blank Model».

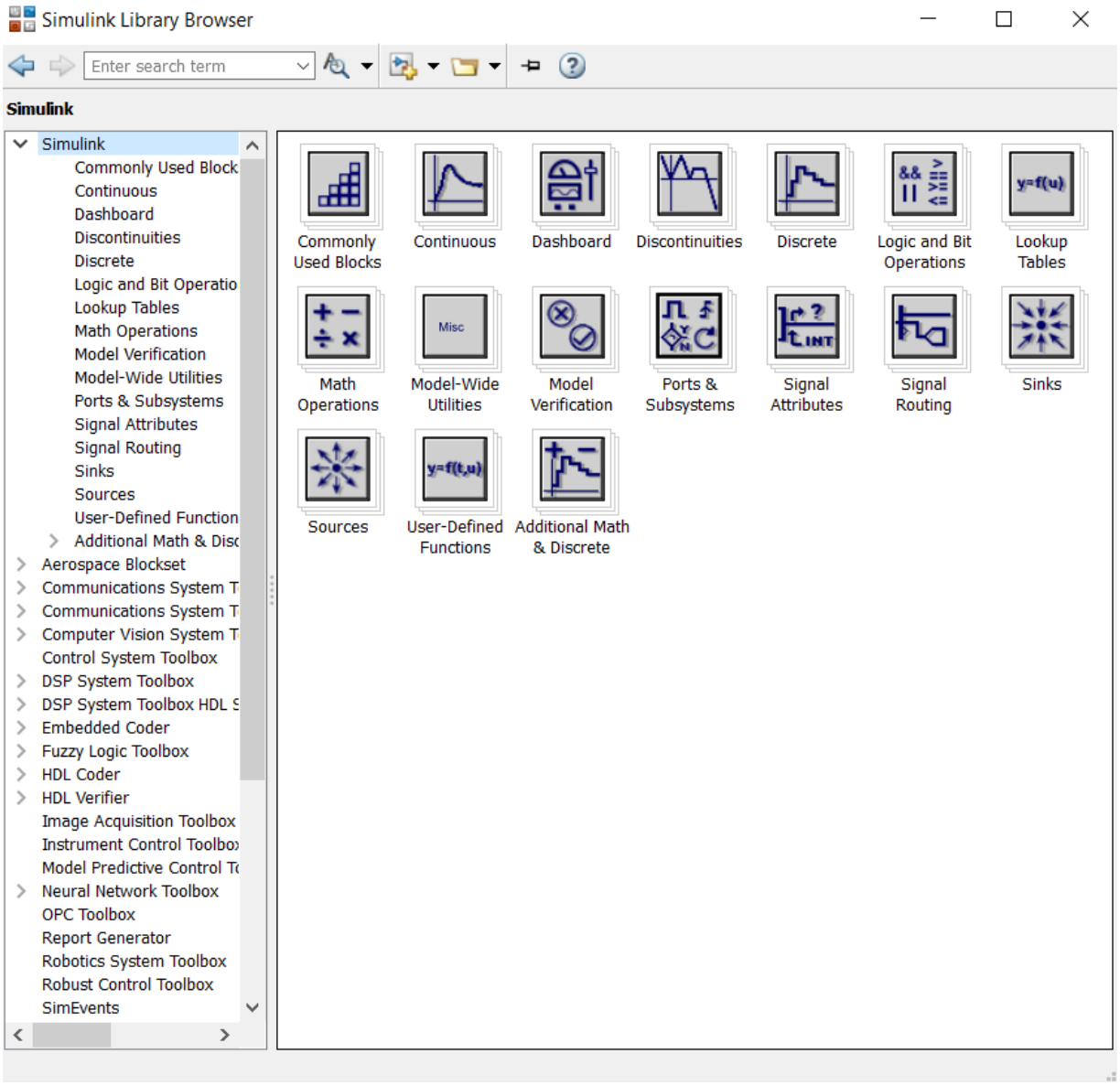


Det vil komme opp en blank modell, som ser slik ut:

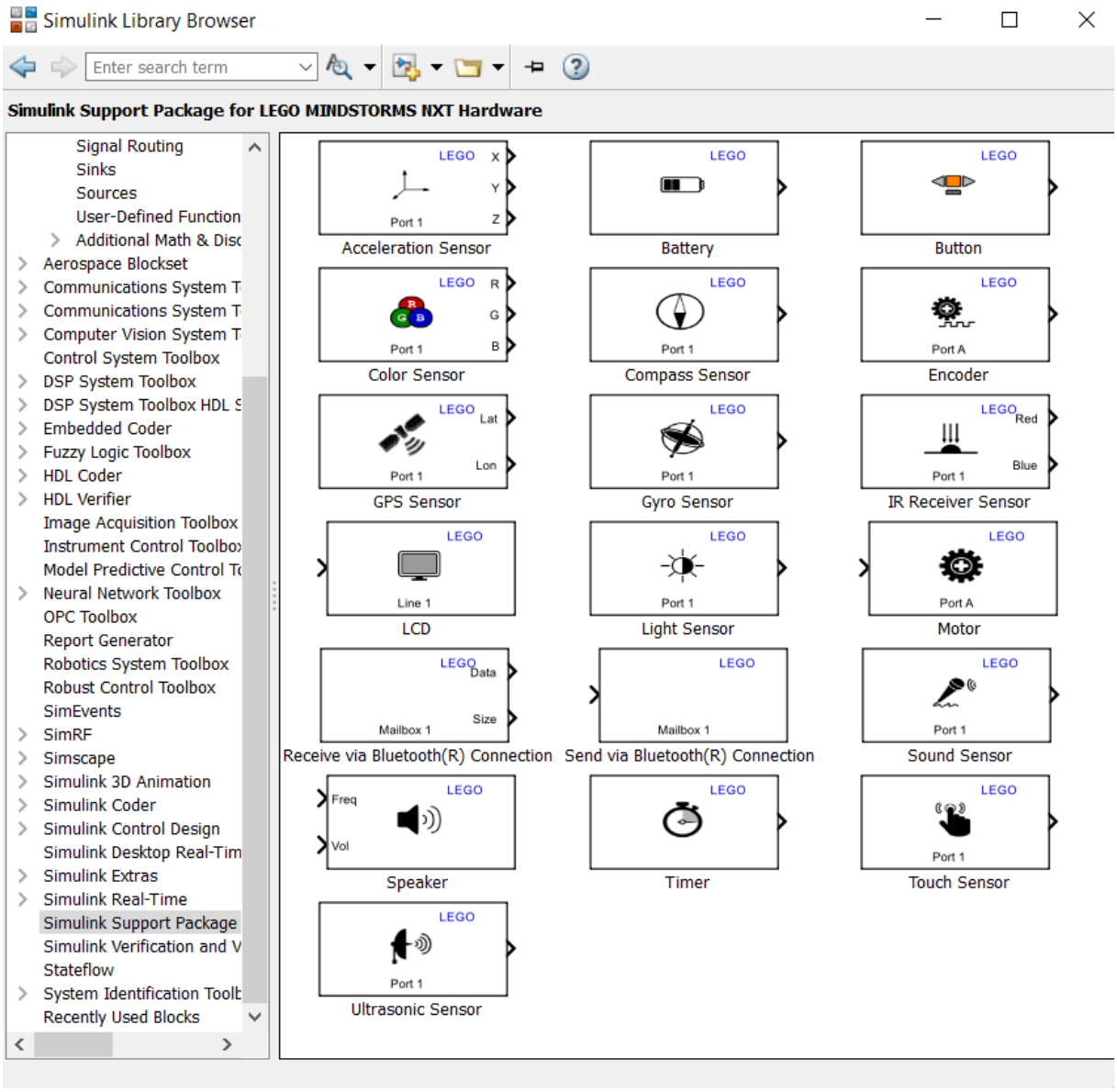


Trykk på  for å få opp «Simulink Library Browser».

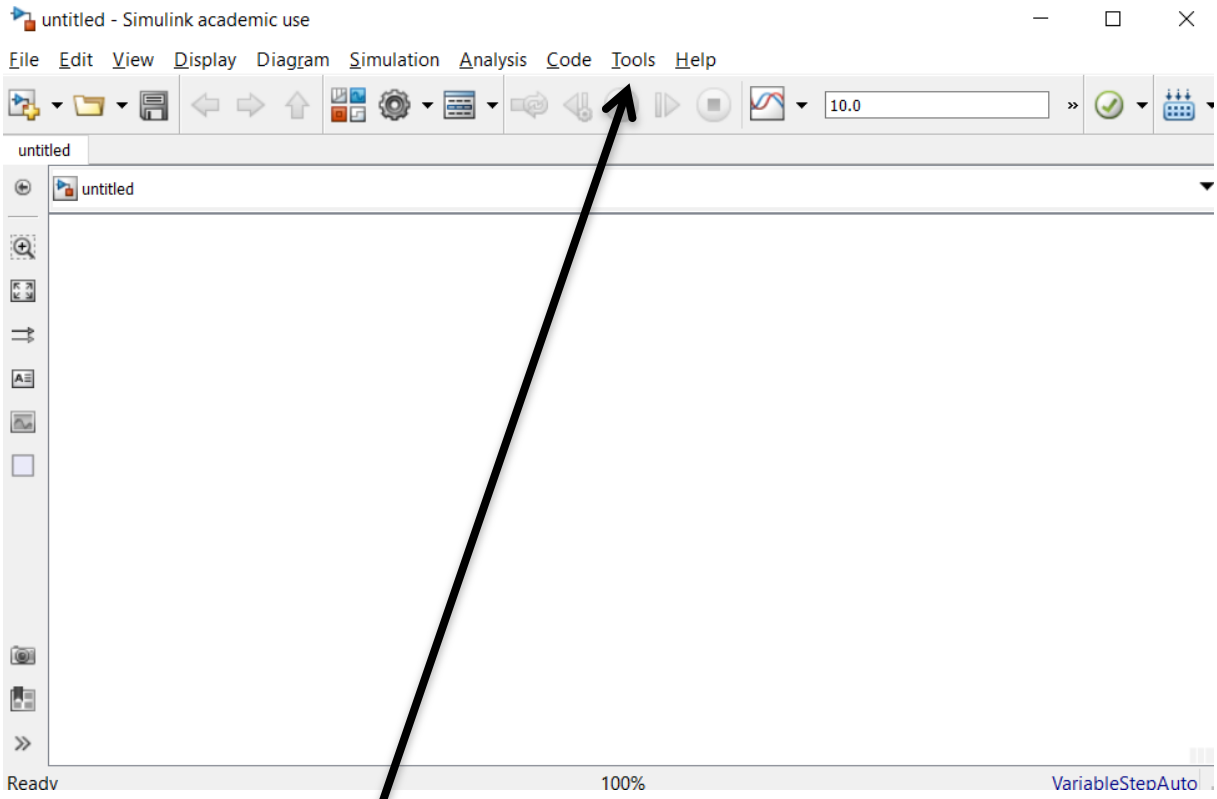
«Simulink Library Browser» er hvor du plukker ut hvilke blokker du vil legge til i modellen. Du drar blokker herfra, og slipper dem inn i modellen.



Blokkene til LEGO MINDSTORMS EV3 finner du under fanen «Simulink Support Package for LEGO MINDSTORMS EV3 Hardware».

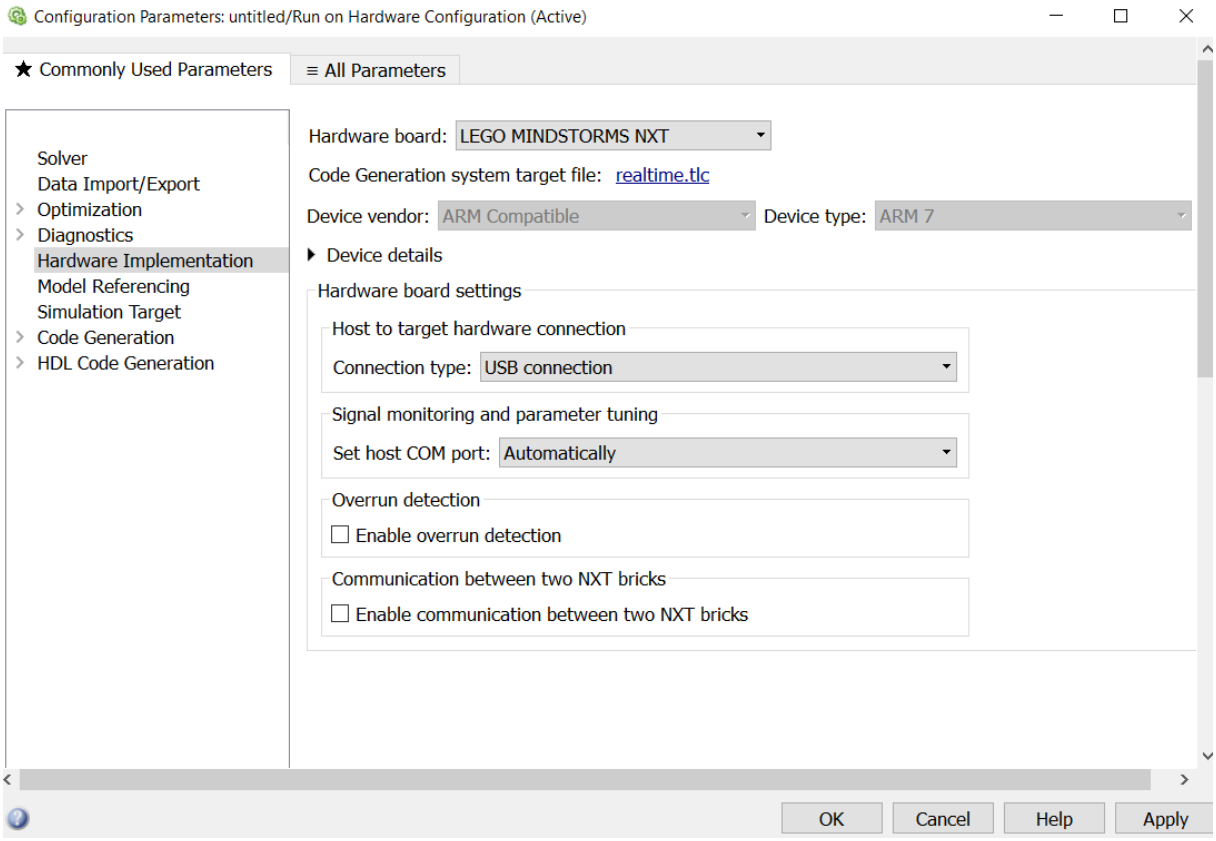


Da kommer dette vinduet opp:

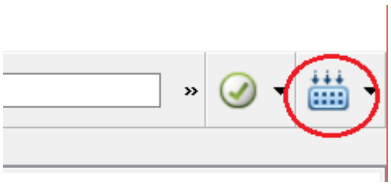


For å gjøre modellen klar til å kunne programmere EV3, må vi gjøre det følgende:

- Lagre Simulink-modellen
- trykk «Tools» HER
- Velg «Run on Target Hardware» «Prepare to Run». Da kommer dette bildet opp:



Pass på at det står «LEGO MINDSTORMS EV3» på «Hardware board» og trykk OK. Dere er nå klar for å lage en modell. For å laste ned modellen trykker dere på denne knappen:

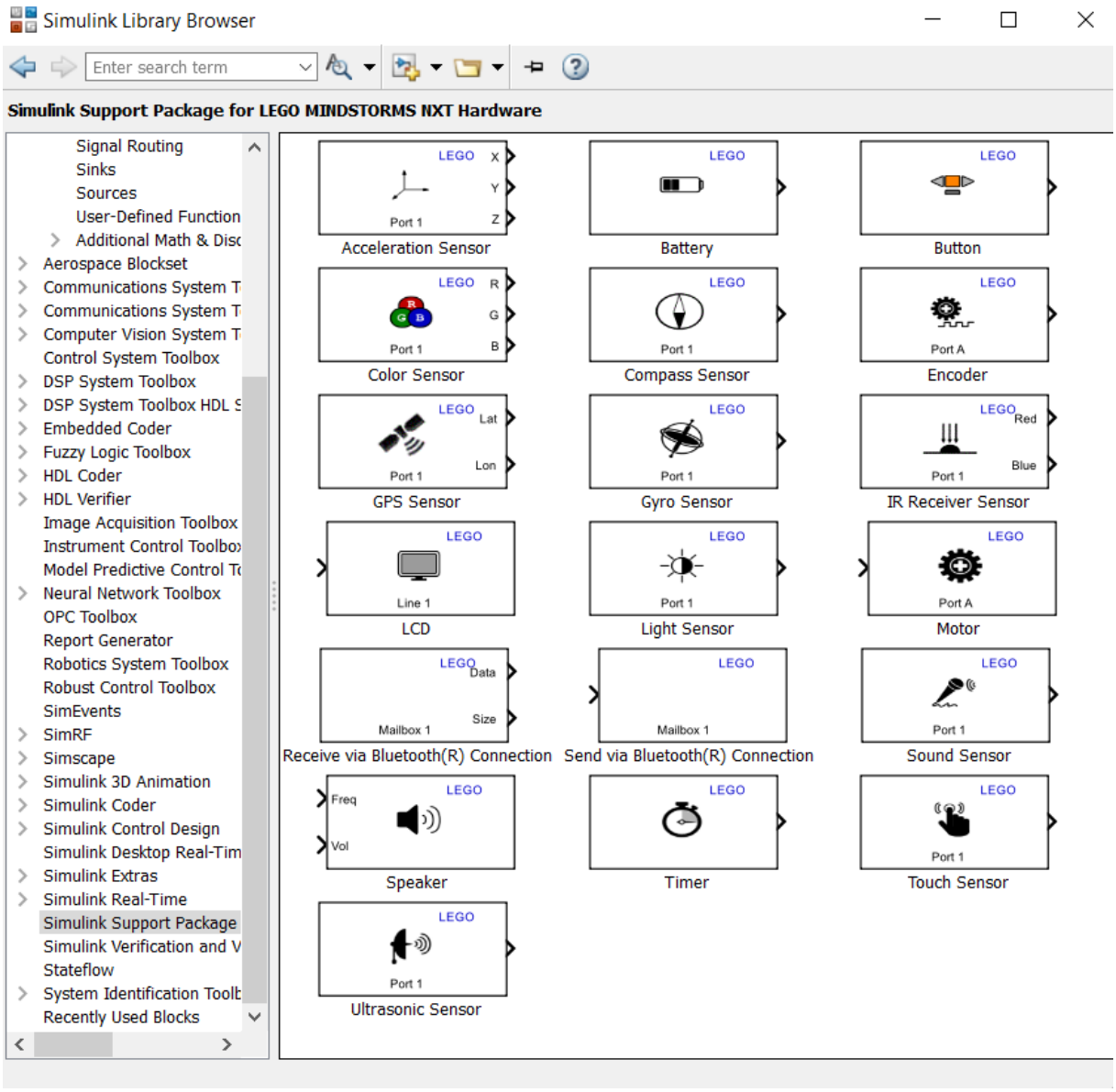


Pass på at EV3 er koblet til, og at ingen programmer kjører på den under opplasting.

Du kan nå lage programmer med Simulink og laste disse opp på EV3. Å lage programmene i Simulink fungerer på dra og slipp-metoden. Det finnes mange blokker som alle har forskjellige funksjoner som kan brukes. Vi kan starte med sensorene og motorene til LEGO.

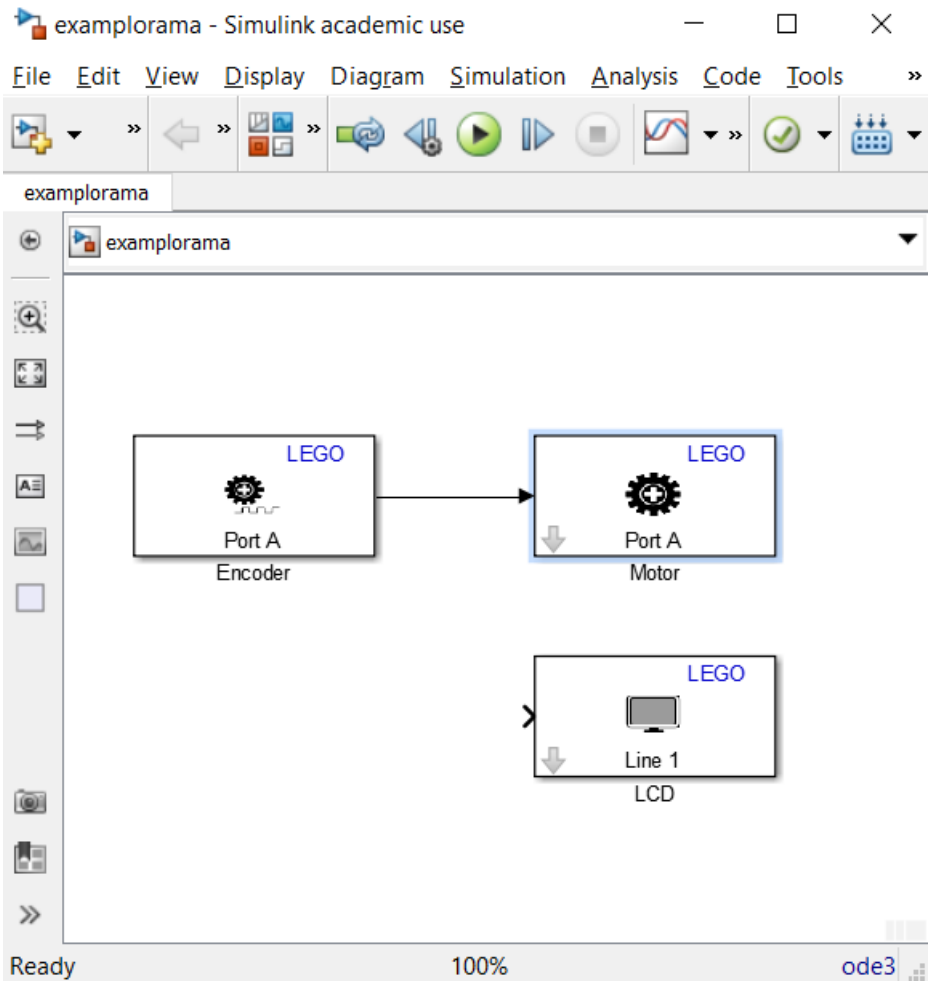
I «Simulink Library Browser» kan dere trykke på «Simulink Support Package for LEGO MINDSTORMS EV3 Hardware».

Her får vi da opp dette vinduet.

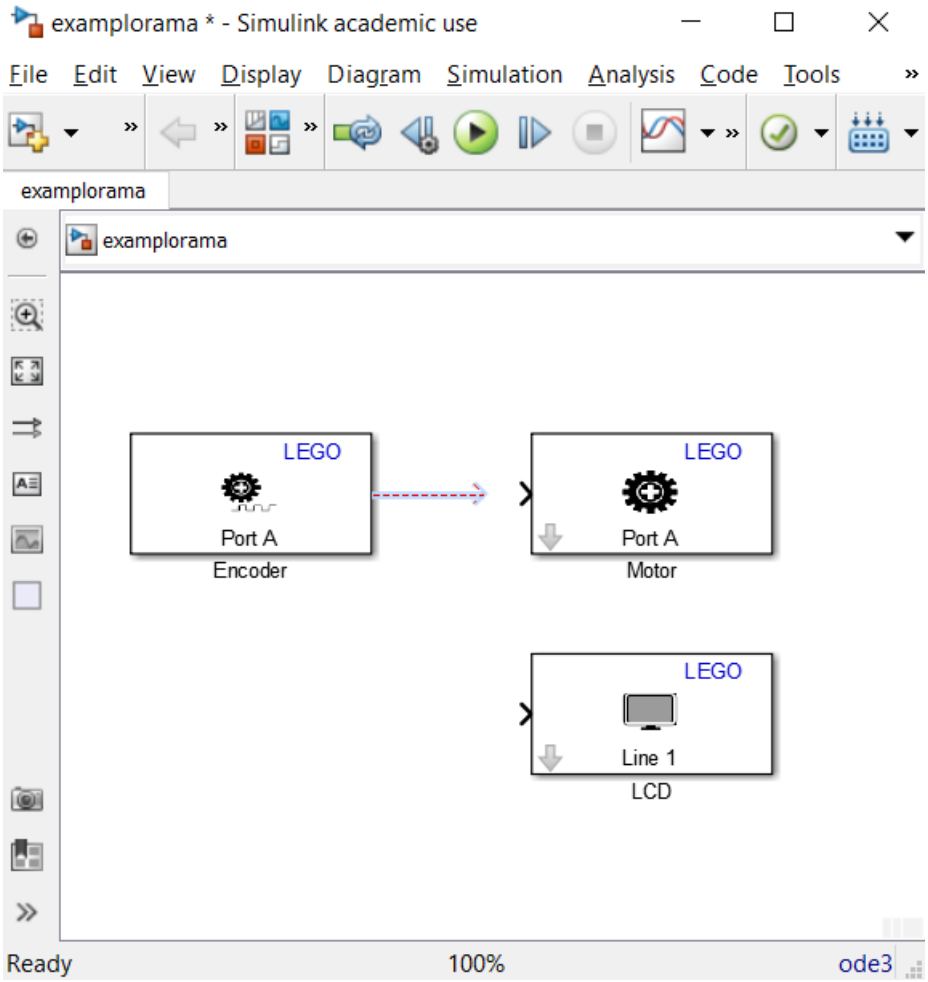


Ta Encoder blokken og dra den over i det for øyeblikket tomme Simulink-vinduet dere åpnet tidligere. Dra så en motor-blokk og en LCD-blokk over. Det skal nå se omtrent slik ut:

Encoderen skal i dette tilfellet ta inn informasjon fra en motor, la oss kalle den Motor 1, om hvor mange grader den har rotert siden programmet startet. Ved å sende dette tallet til en annen motor, si Motor 2, kan man da styre hvor fort Motor 2 skal rotere ved å vri på Motor 1! Trykk nå på > biten til Encoderen med venstre museknapp og hold denne inne, trekk pekeren til > biten av motoren og slipp. Det skal se slik ut:

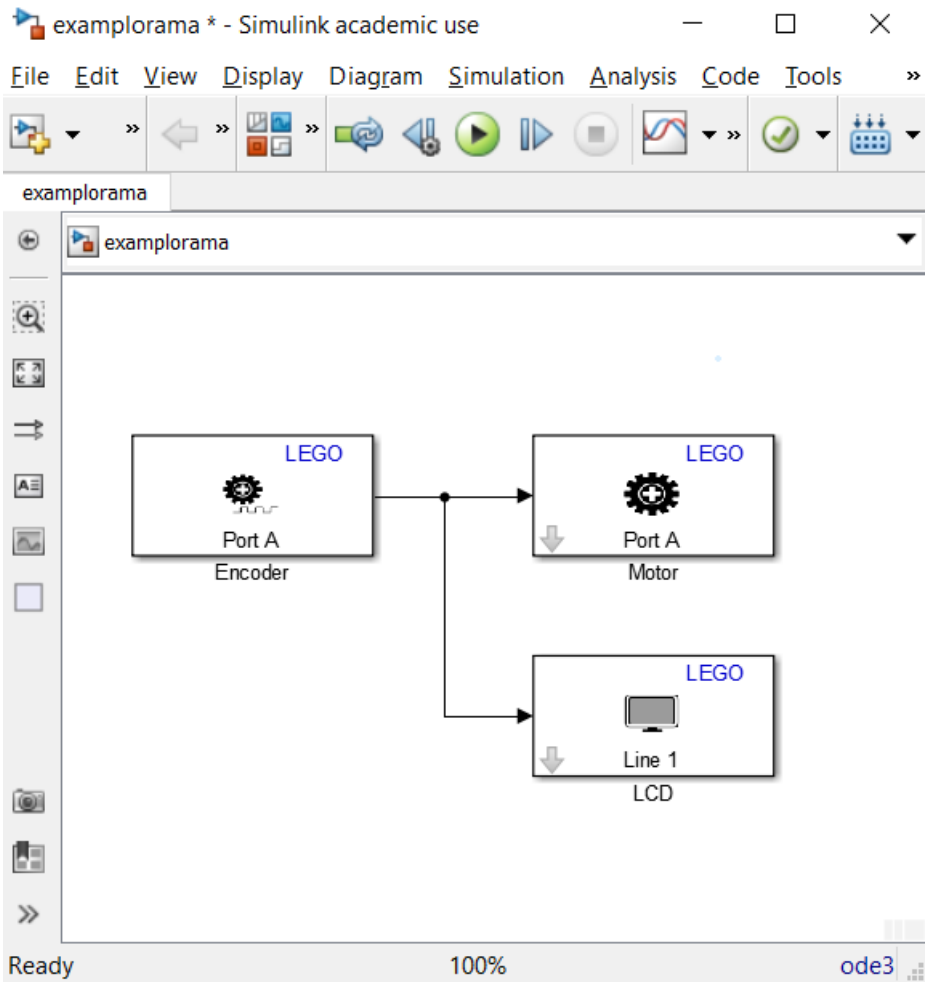

examplorama - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools
examplorama
examplorama
LEGO
Port A
Encoder
LEGO
Port A
Motor
LEGO
Line 1
LCD
Ready
100%
ode3

Verdien fra enkoderen vil nå bli sent til motoren, som så vil snurre rundt. Hvis streken er blitt rødstiplet har det ikke blitt noen kobling.

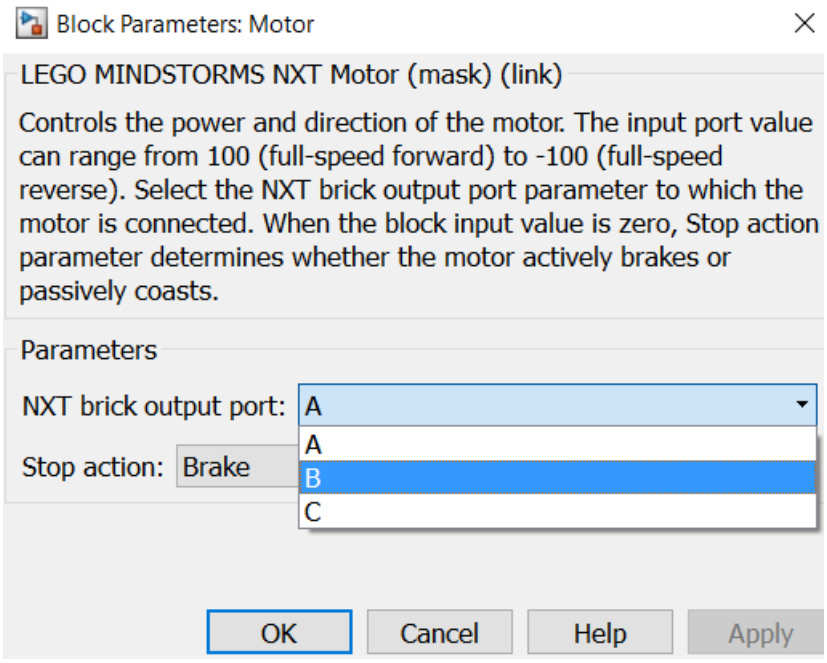


Nå er et godt tidspunkt å lagre på. «file» «save» eller (ctrl + s) lagrer, ikke ha mellomrom eller æøå i navnet.

Koble så > biten til LCD boksen til den svarte linja som går mellom Encoderen og Motoren. Det skal se omtrent slik ut:

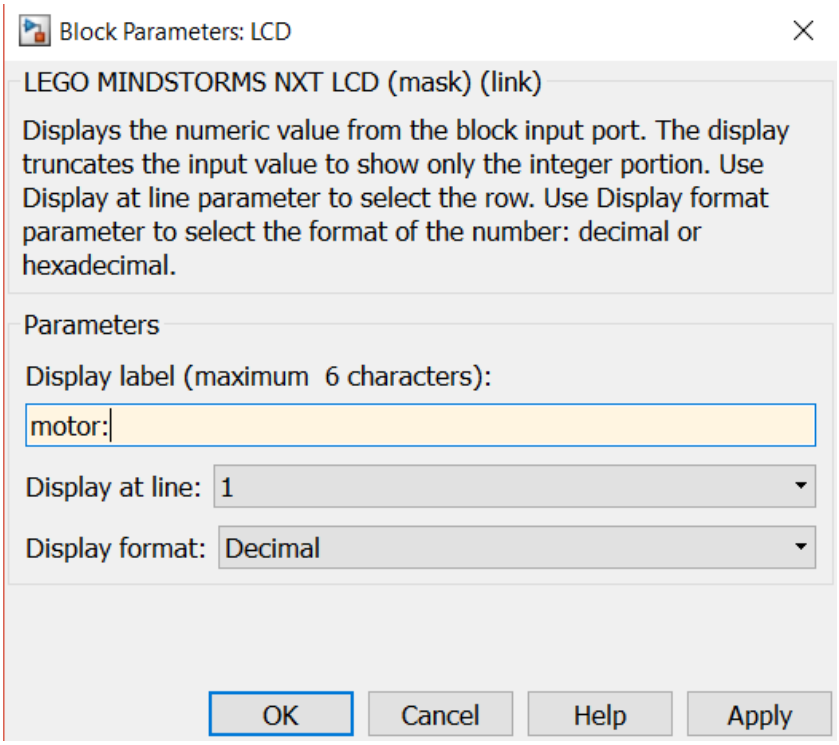


Hvis vi ser etter på Encoderen og motoren ser vi at det står Port A på begge. Det betyr at encoderen i motoren koblet til port A på EV3 styrer motoren koblet til port A. Dobbeltklikk på Motoren og velg port B.



Trykk OK.

LCDen henter også ut verdien fra encoderen, og viser det på linje 1 i displayet. Dobbeltklikk på LCD-boksen og skriv noe fornuftig i «Display label»-boksen (maks 8 tegn).



Trykk OK.

Koble så en motor i inngang A på EV3, og en i inngang B. Koble EV3 opp med datamaskinen og slå den på (trykk på den gule knappen). Last opp programmet dere akkurat lagde på EV3 med hjelp av «Deploy to hardware» knappen.

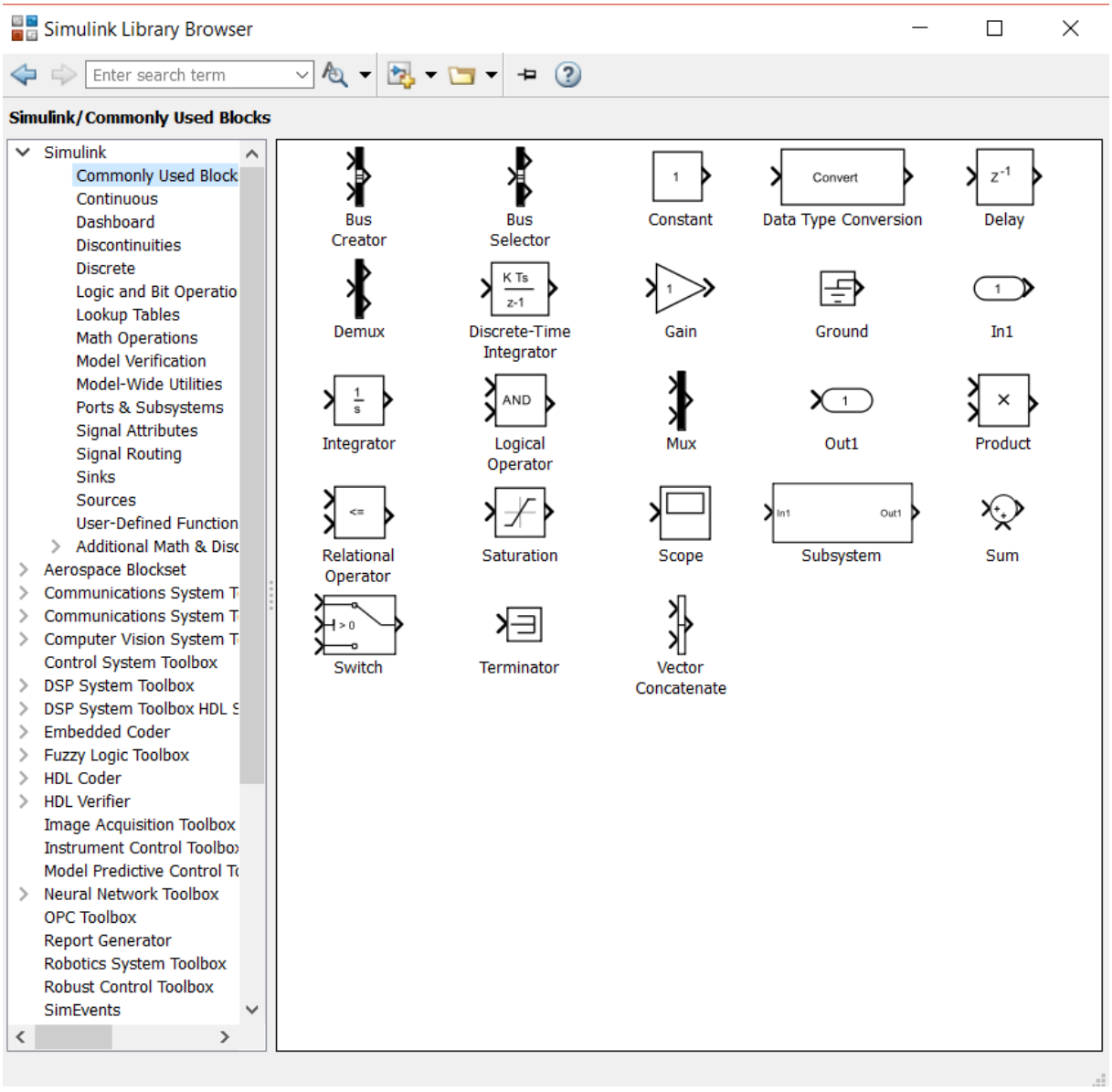
Husk: Før dere kan laste over den nye modellen deres må dere lagre og EV3 må være koblet til datamaskinen. Den kan heller ikke kjøre noen programmer under opplasting.

Prøv å vri på den gule biten på motoren koblet til port A og se hva som skjer med motoren koblet til port B. Følg med på displayet til EV3. Hvilke tallverdier fra encoderen gir maks fart på motoren? For å stoppe programmet på EV3 trykker dere på den grå knappen. Den oransje knappen slår på EV3 og fungerer som enter-knapp i menyene.

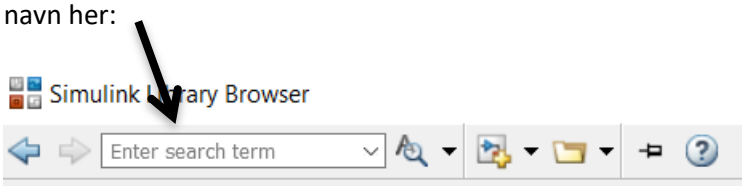
GRATULERER! Dere kan nå lage enkle programmer til EV3. Alle blokkene i «Target for Use with LEGO MINDSTORMS EV3 Hardware» er spesifikke for EV3's innebygde funksjoner, sensorer og motorer. Ellers finnes det i «commonly used blocks» en del andre blokker som dere kommer til å få bruk for.

BESKRIVELSE AV VANLIGE SIMULINK-BLOKKER

Videre følger en beskrivelse av noen av de simulink-blokkene dere kommer til å få bruk for. Blokkene finner du i Simulink Library Browser:



Hvis du ikke finner den blokken du ønsker med en gang kan du søke i søkevinduet etter blokker med navn her:

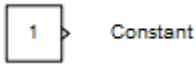


Hvis du lurer på hva en blokk gjør kan man høyre-klikke på den og velge «help» for en utfyllende forklaring på engelsk.

Sum:

- Legger sammen (eller trekker fra) to eller flere tall.

Constant:

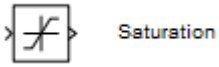


- Gir et konstant tall hele tiden.

Gain:

- Ganger tallet som kommer inn med en konstant

Saturation:

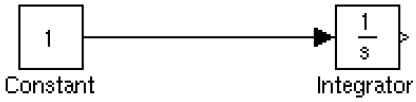


- Metning – setter øvre og nedre grenser for utgangssignalets verdi.
- Hvorfor? Hvis det for eksempel har skjedd en feil (og feil skjer), kan vi nå unngå å overbelaste systemet vårt ved å gi motoren beskjed om å peise på ved en feiltagelse. Fordel.

Integrator:

- Integrerer signalet som kommer inn over tid.

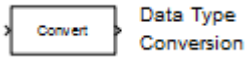
For eksempel:



Her vil integratoren legge til ”1” hver programiterasjon. Etter 1 iterasjon vil utgangen til integratoren være 1, etter 10 iterasjoner 10 osv.

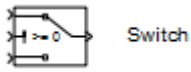
- Protip: Integratoren trenger ingen *saturation* blokk; du kan sette øvre og nedre grenser ved å dobbeltklikke og velge «limit output».

Data Type Conversion:



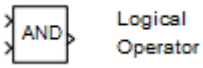
- Konverterer tall fra en *datatype* til en annen.
- Hvis du ikke har noe kjennskap til programmering fra før, ikke fortvil. Du finner snart ut hva i all verden det betyr.
    - Kort forklart kan datamaskiner være ganske strenge, og de forskjellige delene av programmet vil helst ha tallene sine i et format de forventer (i.e. hvis det vil ha heltall må du gi dem heltall, hvis det vil ha desimaltall … etc). Denne blokken fikser det for deg.

Switch:



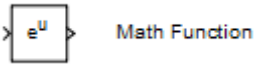
- Sammenligner inngangssignalet med et tall, og gjør noe hvis det er større-enn-eller-lik, noe annet hvis det er mindre.

Logical Operator (AND):



- Gir verdiene 1 eller 0 (sant/usant) ut avhengig av inngangsverdiene. Hvis alle disse er 1, blir utgangen 1. Hvis en eller flere av dem er 0, blir utgangen 0 («den OG den OG den OG…»).
- Samme blokk kan gi andre logiske funksjoner (for eksempel «den ELLER den ELLER…»).

Math function:



- Gir en rekke forskjellige mattefunksjoner som kan velges i menyen som kommer opp når du dobbeltklikker på blokken.
- Blant annet finnes *modulo*, en operasjon som tar to tall inn og gir ut *resten* du får når disse deles på hverandre. 5 mod 2 = 1. http://en.wikipedia.org/wiki/Modulo_operation

EKSEMPEL 1 – CRUISE CONTROL[1]

Dere legger kanskje merke til at motorene ikke roterer med samme hastighet på samme pådrag? Dette kan skyldes veldig små forskjeller i konstruksjonen, og kan skape store problemer når man skal prøve å kjøre en rett strekning. Skulle ønske det var en måte å få motorene til å rotere nøyaktig en rotasjon i sekundet! Flaks at vi har en super kybernetisk løsning på dette problemet! Motorene har en vinkelmåler innebygd, og denne kan vi bruke til å lage en salgs *cruise control*; ikke ulikt det man finner på biler! Hvis man mens programmet kjører sammenlikner informasjonen fra vinkelmålerne (den *faktiske* rotasjonshastigheten) med den *ønskede* rotasjonshastigheten, i dette tilfellet 1 rotasjon i sekundet, kan man regulere pådraget slik at rotasjonshastigheten blir slik man ønsker. En enkel måte å gjøre dette på er å regne ut pådraget med følgende formel:

$$Pådrag = e * K_p$$

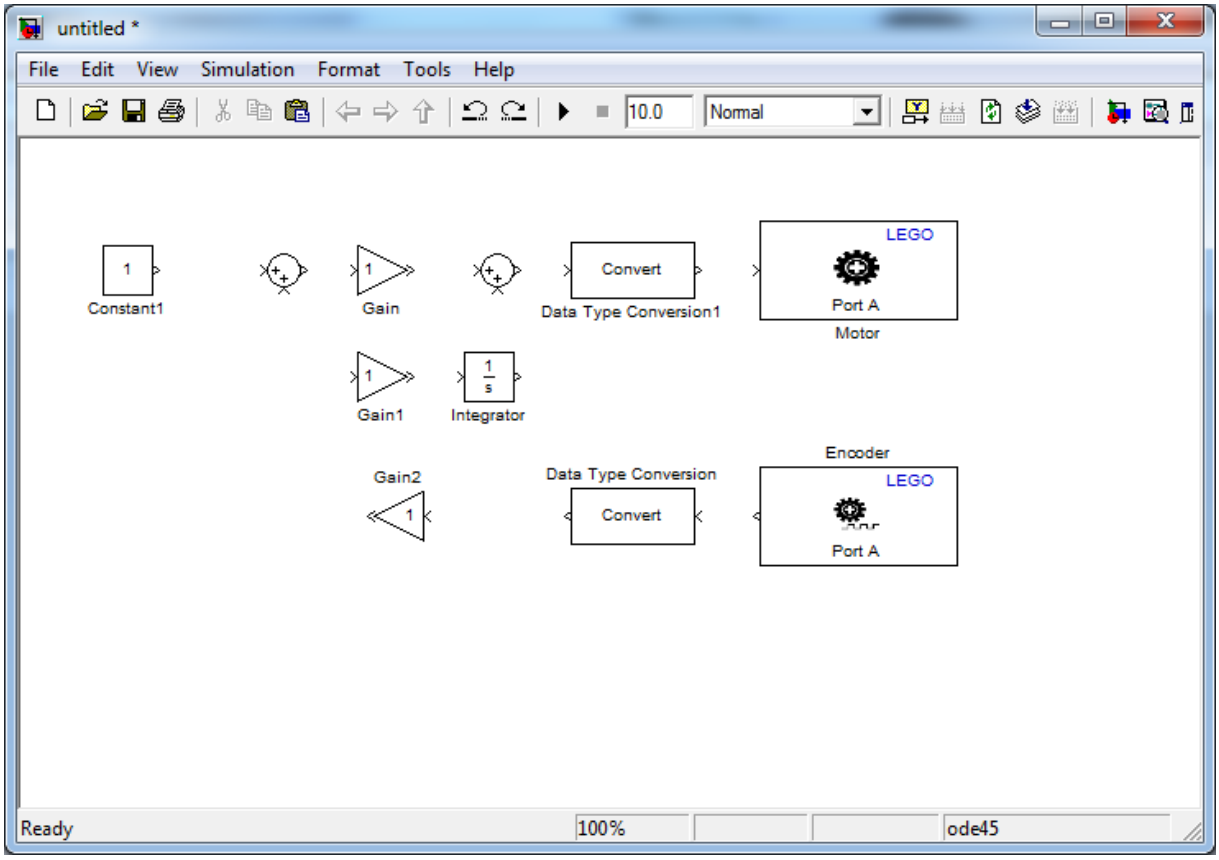
Her er *e = error = (ønsket hastighet – faktisk hastighet)* og K_p er en eller annen konstant (p for Proporsjonal). Her ser vi at pådraget blir større jo større avviket er, og blir mindre og mindre ettersom avviket reduseres. Dette er en enkel reguleringsalgoritme, som dessverre er litt *for* enkel. Den vil gi et *stasjonært avvik* (konstant feil) mellom ønsket og faktisk hastighet (kan du se hvorfor?). Derfor utvider vi regulatoren vår ved å legge til integralet av avviket over tid. Dette vil i praksis legge til summen av alle avvikene over tid, noe som vil føre til at pådraget øker så lenge det er et positivt avvik, helt til det ikke er noe avvik lenger! Nå ser formelen for regulatoren vår slik ut:

$$Pådrag = e * K_p + K_i * \int_0^t e\, dt$$

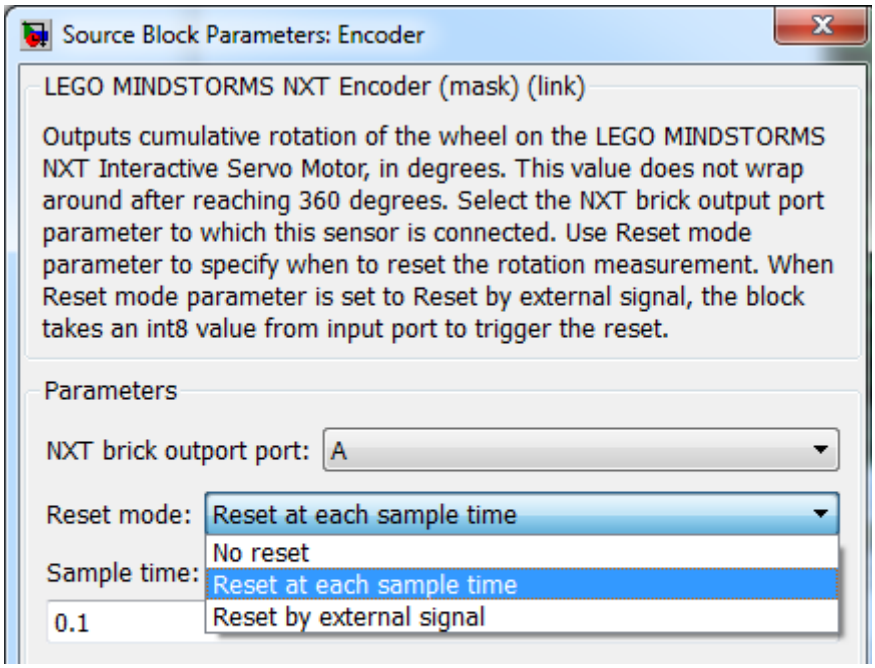
der K_i er en ny konstant (i for Integral). En sånn regulator kalles en PI-regulator, og fungerer fint til vårt formål. Ikke bli redd hvis du ikke skjønner hvordan dette virker, men ta gjerne en prat med din læringsassistent om regulatorer! Dette er *ikke* siste gang du kommer til å høre om dette!

[1] Dette eksempelet er lagd i en eldre versjon av Simulink, men bortsett fra hovedsakelig kosmetiske forskjeller er meste det samme som i 2016a.

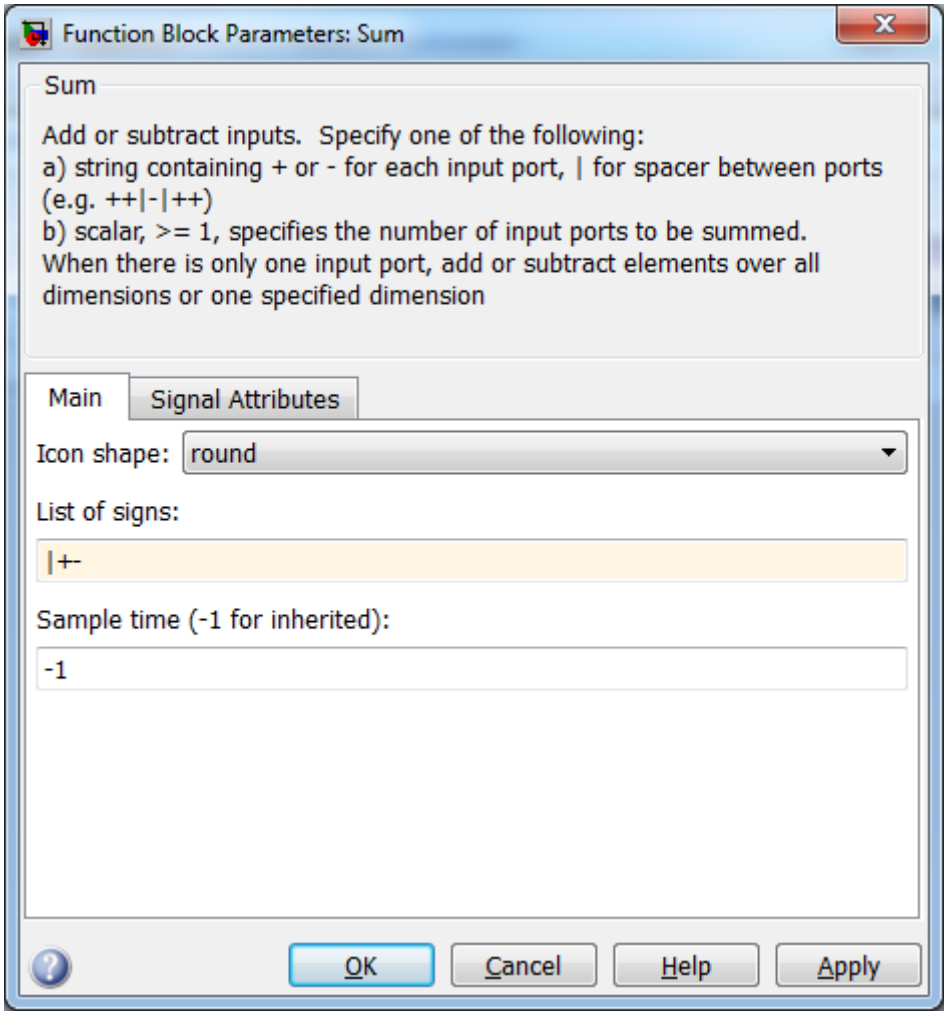
Nå skal vi prøve å realisere regulatoren vår! Lag først en ny Simulink fil. Dra så disse elementene inn i den:



Encoderen må vi endre fra å gi ut vinkelendring siden programmet startet til å fortelle oss hvor mye den har endret seg siden forrige måling. Det gjøres ved å dobbeltklikke på den og endre ”reset mode” til ”Reset at each sample time”:



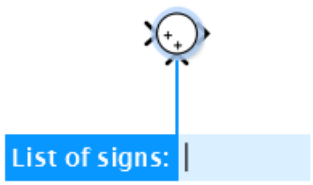
Gå så inn på den første *Sum*-blokken fra venstre på forrige side og endre den til dette[2]:

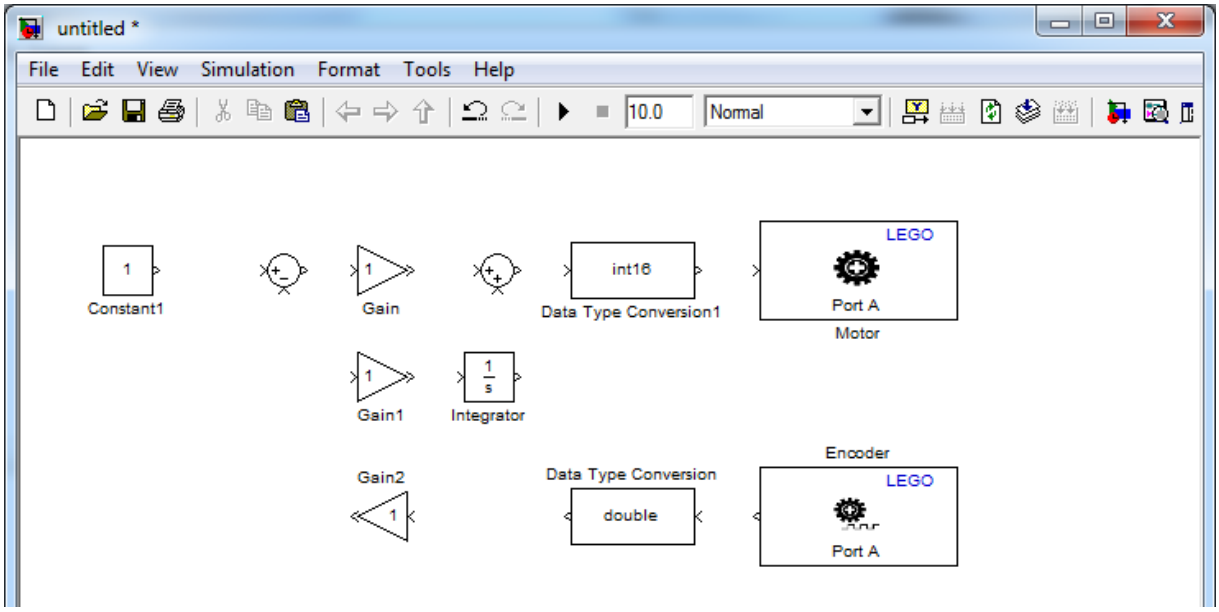


Det gjør at blokken trekker fra den ene inngangen, slik at vi får (*a–b=ut)* istedenfor (*a+b=ut)*. Dette er **meget viktig**. Hvis vi ikke gjør det får vi såkalt *positiv tilbakekobling*, og det kan (og VIL) forårsake at roboten vår krasjer spektakulært! (Hvorfor?)

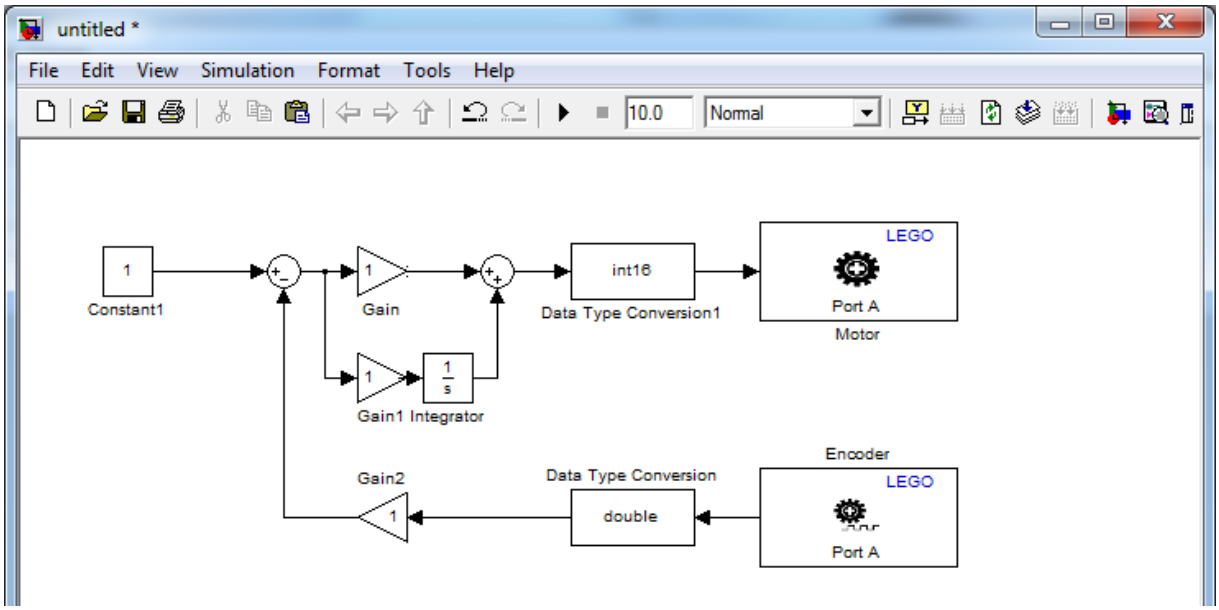
Nå kommer også den mystiske *Data Type Conversion*-blokken inn. Integratoren funker ikke på verdier av typen *int (heltall)*, som er det enkoderen gir ut, så disse må settes til mer integrator-vennlig *double (desimaltall)*. Motoren på sin side benytter ikke *double*, så her må vi konvertere til *int16* (som er nesten det samme som *int* men benytter flere siffer). Når du er ferdig skal det se slik ut:

[2] I 2016a har du også ofte mulighet til å sette sentrale blokk-parameter med en gang du trekker blokken inn i modellen:





Deretter kobler vi opp alt slik:



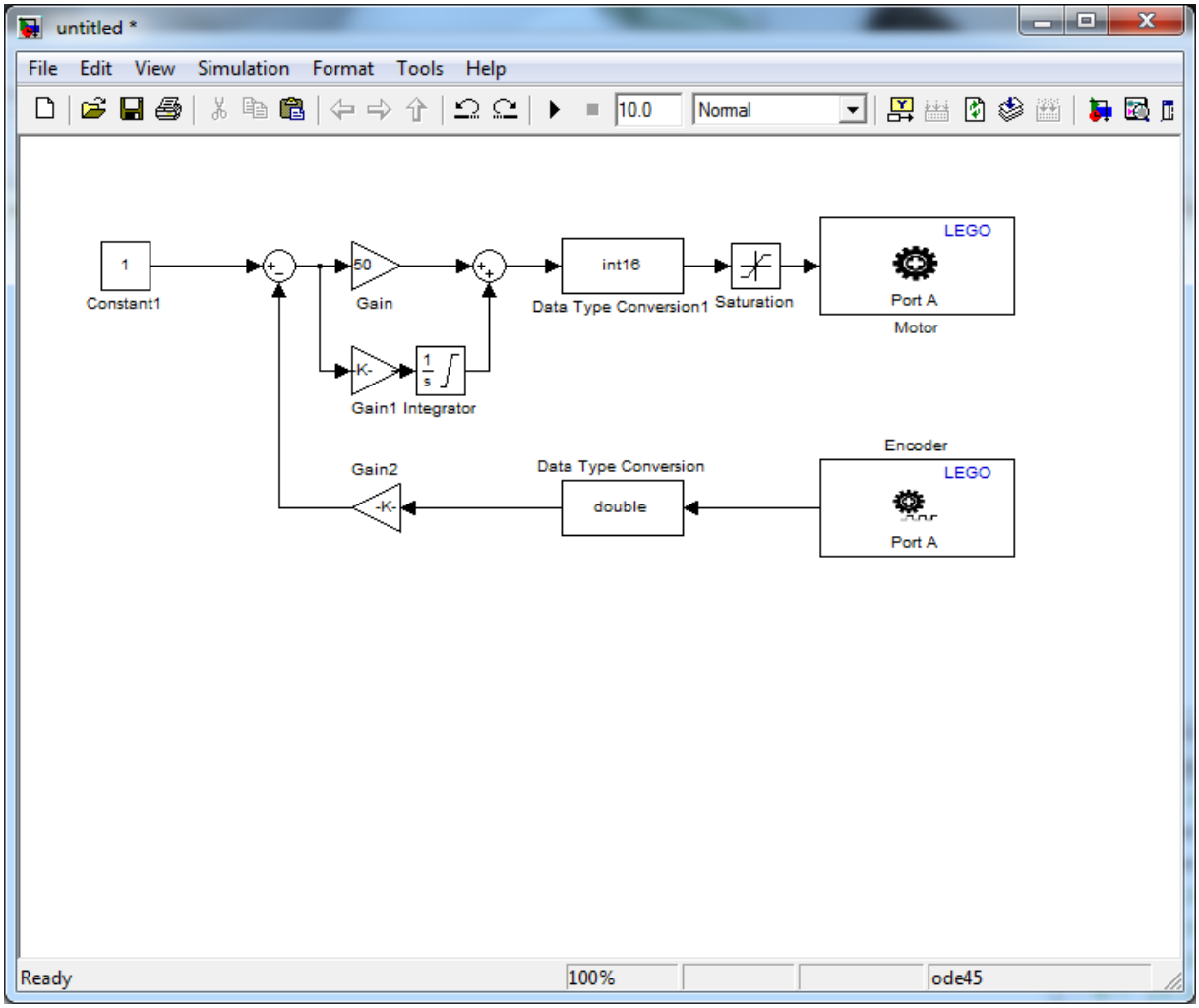
Prøv å sammenlikne modellen med formelen for pådraget beskrevet på side 16. Nå er det på tide å sette inn verdier.

Konstanten 1 er en *referanse* for regulatoren vår, altså vår ønskede hastighet, og gjør at systemet vil prøve å holde 1 rotasjon i sekundet.

"Gain" og "Gain1" representerer K_p og K_i, og kan settes til henholdsvis 50 og 100, men det kan hende at dere finner bedre verdier ved litt eksperimentering. "Gain2" derimot settes til 1/36.

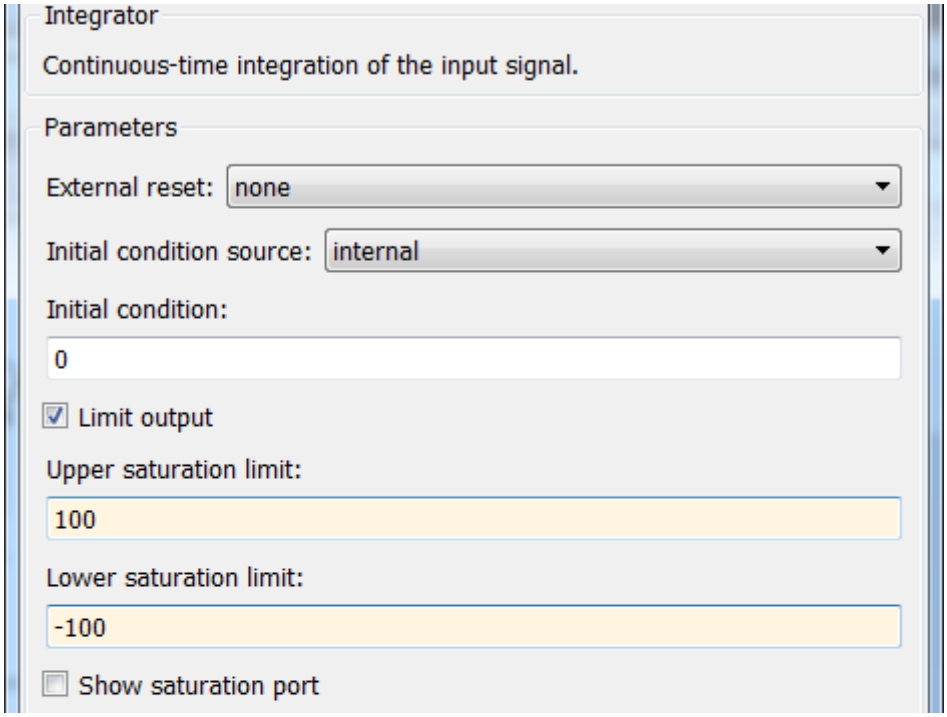
Grunnen til dette er at vi er interessert i *rotasjoner per sekund*, men encoderen gir *grader siden sist*, altså må vi dele på 360. I tillegg rapporterer encoderen 10 ganger i sekundet, så vi må gange med 10. altså: 1/360 (grader til rotasjoner) * 10 (sjekker 10 ganger i sekundet).

Hvis vi i tillegg legger til en begrensing på integratoren slik at den bare kan ha verdier mellom -100 og 100 ved å sette en *Saturation* mellom int16 blokken og motoren med en begrensning på -100 til 100 blir det seende slik ut:

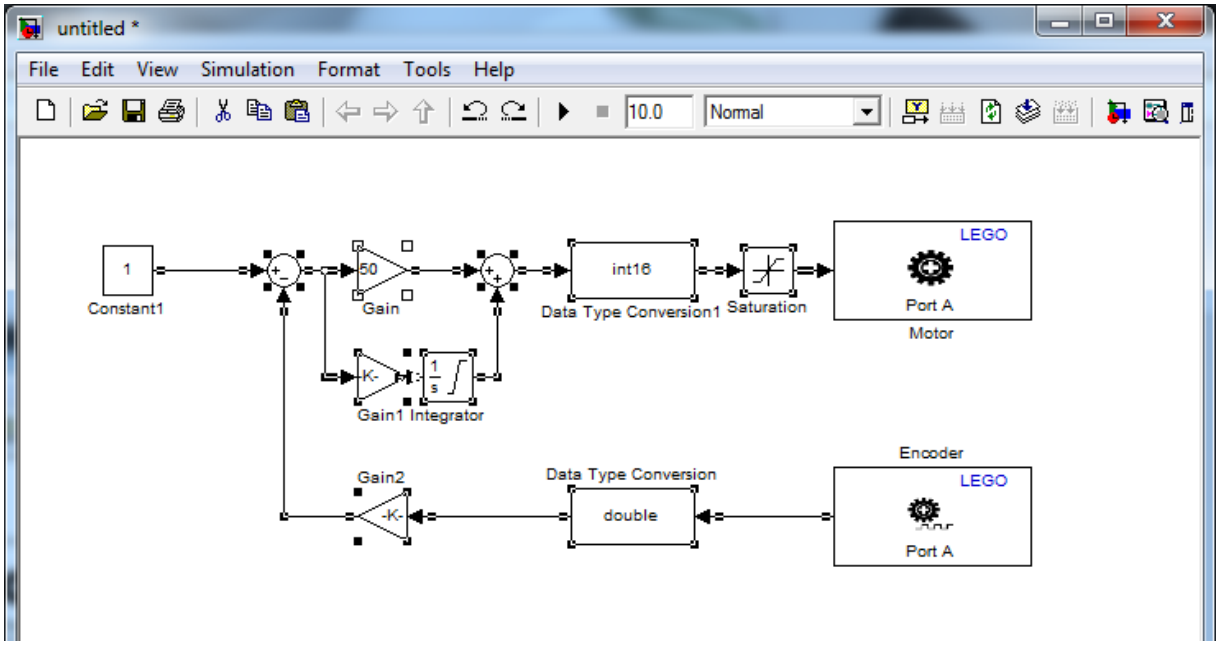


Enkle regulatorer av denne typen er svært sentrale i kybernetikken. Diskuter med læringsassistenten deres hvordan denne virker, og hvorfor!

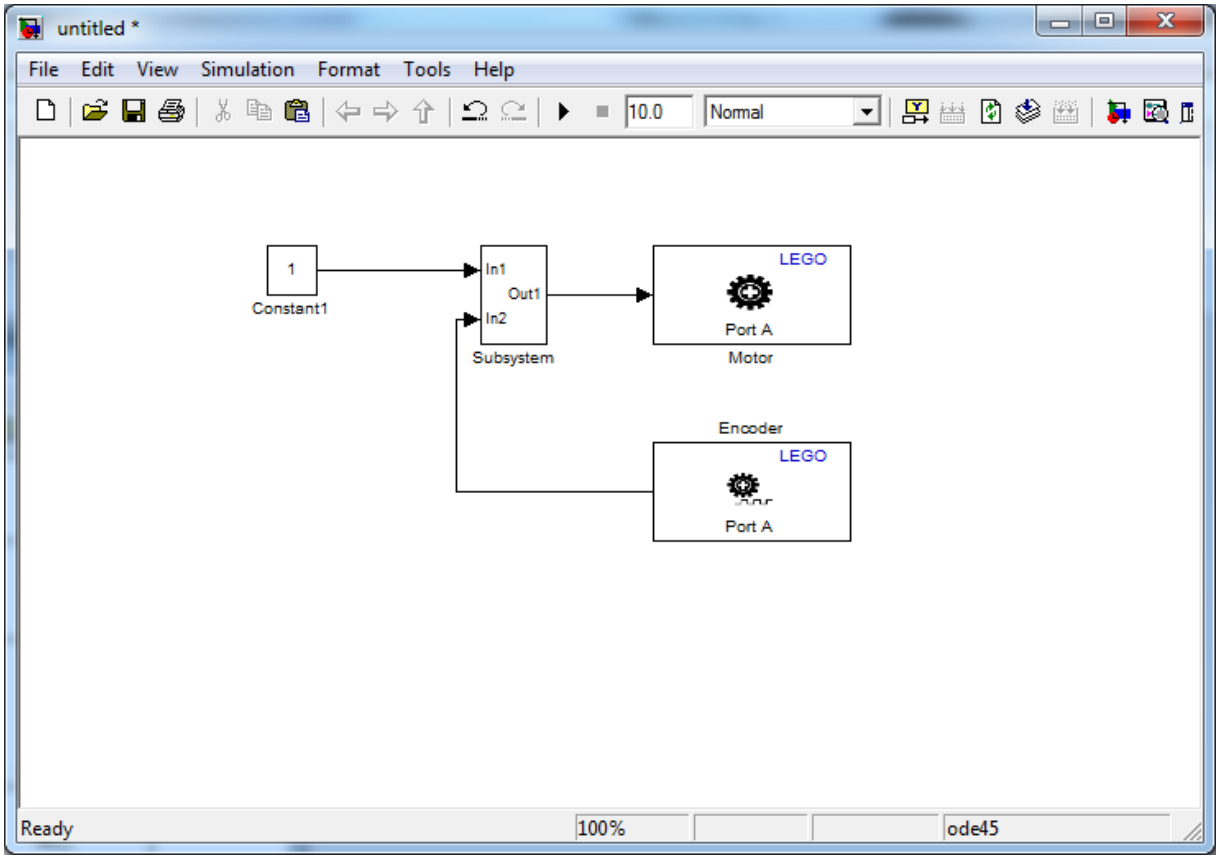
Begrensing på integratoren gjøres slik:



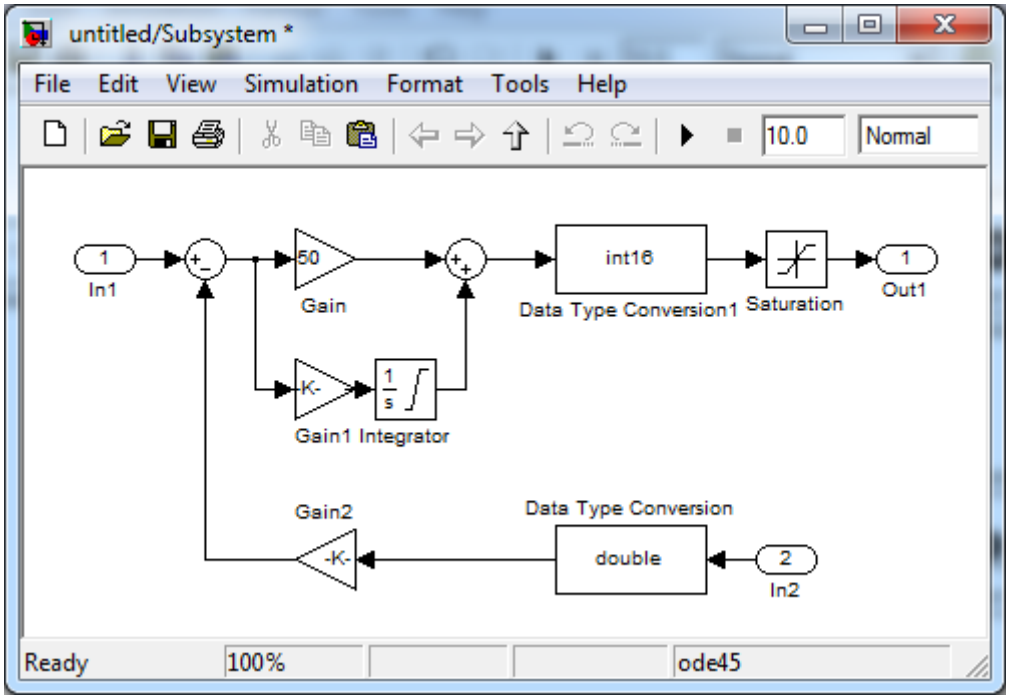
Det ser litt rotete ut på bildet over, derfor kan vi med fordel rydde litt. Merk alt som er merket på dette bildet:



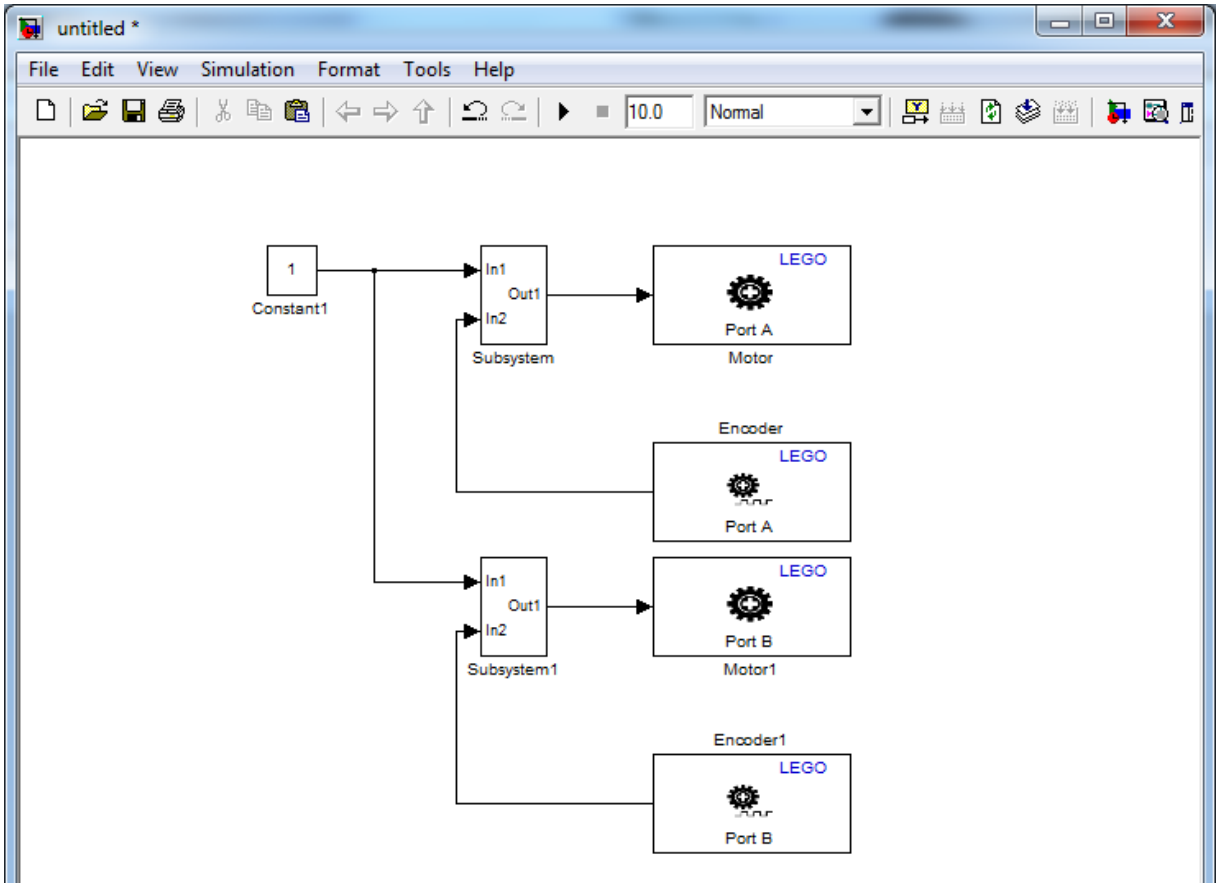
Høyreklikk og velg ”create subsystem from selection”. Du vil da få noe som ser slik ut:



Trykker du på ”Subsystem” kommer alt du merket opp:



Nå kan vi kopiere vårt nye subsystem og lage motorstyring for to motorer:

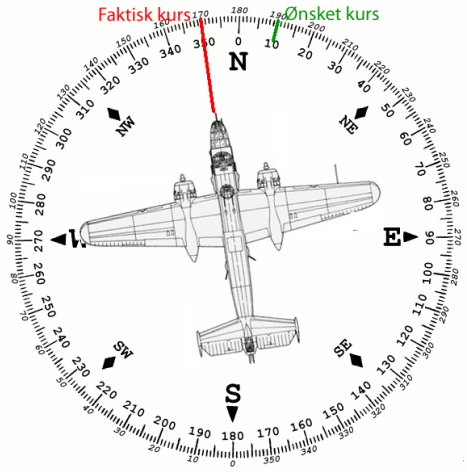


Dette programmet vil sørge for at roboten kjører rett frem, samtidig som den kan svinges ved å endre *referansene* til en av de to motorene (her er den konstant 1 til begge).

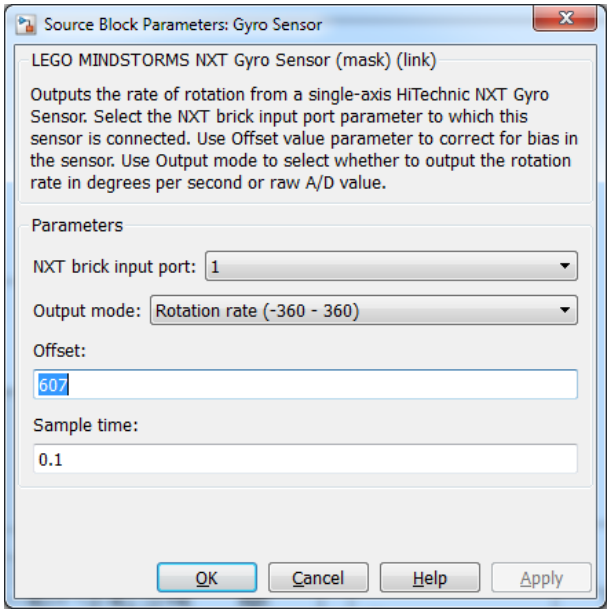
EKSEMPEL 2 – NAVIGERE MED GYROSKOP

Et gyroskop er en sensor som måler *vinkelhastighet* rundt en gitt akse. Vi antar at vi ønsker å styre *vinkelen* til roboten. Hvordan kan vi bruke gyroen til å finne vinkelen (hint: hva skjer når man integrerer opp hastighet med hensyn på tid)?

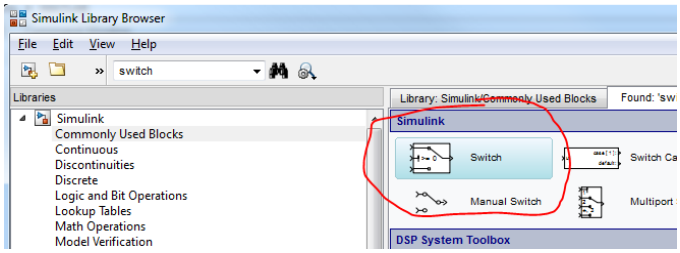
Når dere har funnet en måte å finne vinkelen til roboten på, kan denne verdien brukes til å holde styr på retningen når dere navigerer gjennom hinderløypa. La oss si at dere har lyst til å holde en kurs på 10 grader. Når vi så navigerer i hinderløypa, bruker vi vinkelen til å kontinuerlig sammenlikne robotens *faktiske* kurs med *ønsket* kurs (10 grader). Avviket *e* blir da forskjellen mellom de to. Jo større dette avviket er, jo mer ønsker vi å svinge oss inn mot ønsket kurs. So far so good...?



Det er noen utfordringer ved å bruke en gyro. Den er blant annet utsatt for *bias*. Det vil si at den gir en verdi forskjellig fra null selv om den ligger helt i ro. Hva skjer når man integrerer opp en konstant verdi (bias)? For å kompensere for dette kan man endre *offset* i parameterinnstillingene til gyro-blokka i Simulink. Prøve å lese verdien fra gyroen rett ut på skjermen og endre offset til målingen viser null. Det er viktig at gyroen ligger i ro mens dere tar disse målingene.



En annen utfordring med gyroen skyldes støy. Selv om gyroskopet er kalibrert vil det være støy i målingene. Dette kan observeres ved at målingene fra gyroskopet varierer selv når gyroskopet ligger stille (se på målingene fra gyroskopet på LCD skjermen). Denne støyen blir forsterket når bilen kjører. Hva tror dere dette kan ha å si når man integrerer opp verdien for å oppnå et mål på vinkelen? For å hjelpe til på dette problemet kan man legge inn en *dødsone*. Det vil si at man ikke integrerer opp vinkelen hvis ikke gyromålingene er over en viss terskelverdi. For å oppnå dette kan man ta i bruk en *switch*-blokk som ble nevnt tidligere. Det finnes også flere løsninger.



Denne blokken har tre innganger. Den midterste inngangen styrer switchen, og bestemmer om den nederste inngangen eller den øverste inngangen skal slippe ut gjennom utgangen. Ved å dobbeltklikke på blokken får dere opp innstillinger og kriterier for hvordan den skal trigges.

EKSEMPEL 3 – TERMINALHASTIGHETEN TIL EN FALLSKJERMHOPPER

Dette eksempelet er ikke så relevant for LEGO, men er et fint eksempel på hvordan det er å bruke MATLAB/Simulink til å beregne fysiske fenomener med utgangspunkt i differensiallikninger.

Fallskjermhopperen Kari lurer på hvor fort hun faller mot bakken når hun har oppnådd likevekt mellom tyngdekraften og luftmotstanden (dette kalles terminalhastigheten). Luftmotstanden er proporsjonal med hastigheten i annen, og kan skrives $F_d = Kv^2$, der *K* er en konstant (kalles drag-konstanten). Tyngdekraften kan skrives $F_g = -mg$. For å løse problemet setter hun opp Newtons berømte lov

$$\sum F = ma$$

$$F_g + F_d = ma$$

$$-mg + Kv^2 = ma$$

Ved å dele på m får vi:

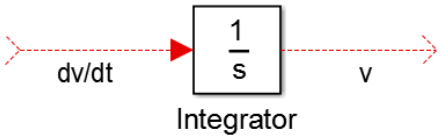
$$a = \frac{K}{m}v^2 - g$$

som kan skrives

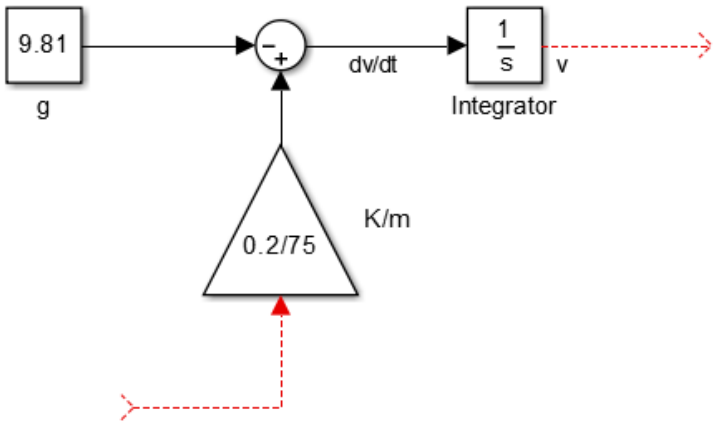
$$\frac{dv}{dt} = \frac{K}{m}v^2 - g$$

Dette er en differensiallikning som beskriver sammenhengen mellom akselerasjon og hastighet, og det er ut fra denne mulig å finne ut hvordan hastigheten endrer seg med tiden. Dette kan gjøres matematisk, eller ved å simulere i for eksempel Simulink. Først må konstantene i likningen bestemmes. Tyngdekraften *g* er som kjent 9.81 [m/s^2], *K* er for et menneske cirka lik 0.2, og Kari veier 75 kg med alt utstyret på (*m* = 75 [kg]). Da kan vi begynne å implementere dette i Simulink!

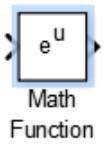
Lag en ny modell og dra inn integratorblokken fra Simulink Library Browser. Tanken er at hvis $\frac{dv}{dt}$ kommer inn på inngangen, kommer *v* ut på utgangen:



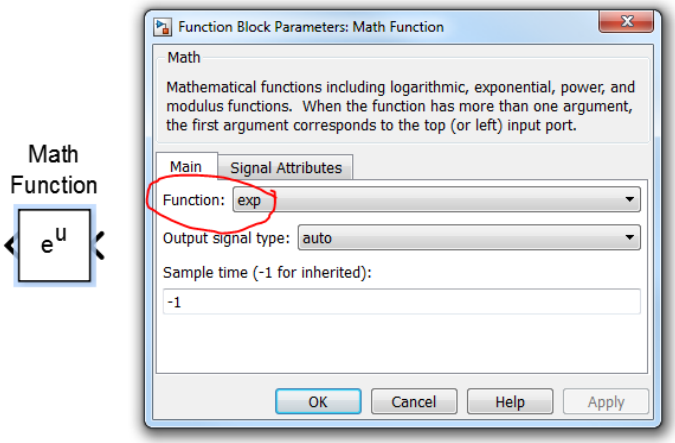
Vi vet fra differensiallikningen over at $\frac{dv}{dt} = \frac{K}{m}v^2 - g$ så la oss få inn dette på inngangen til integratoren. Dra over en Constant-blokk, en Sum-blokk og en Gain-blokk, og ordne dem slik:



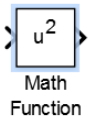
For å rotere en blokk, sånn som det er gjort med Gain, kan man markere blokken og trykke Ctrl+R. Inne i Gain-blokken kan du skrive 0.2/75, som da blir $\frac{K}{m}$. Ser du hvordan dette representerer differensiallikningen vi kom fram til? Nå er vi nesten i mål. Det neste vi må gjøre er å finne v^2. For å få til det, gjør vi et lurt triks: vi "stjeler" *v* fra utgangen av integratoren og opphøyer den i annen med Math Function-blokken:



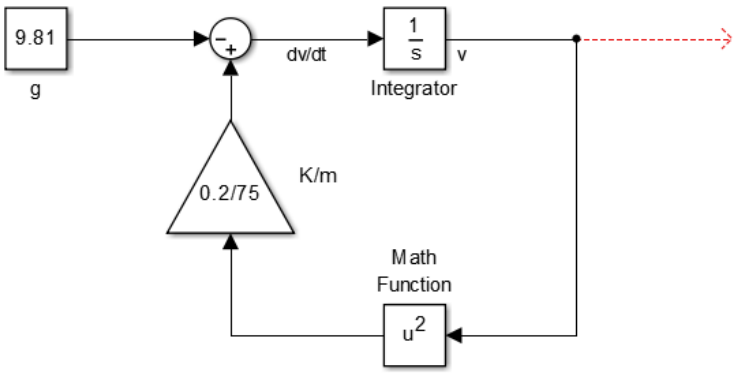
Dra en Math Function-blokk over i modellen din og dobbeltklikk på den slik at dette vinduet dukker opp:



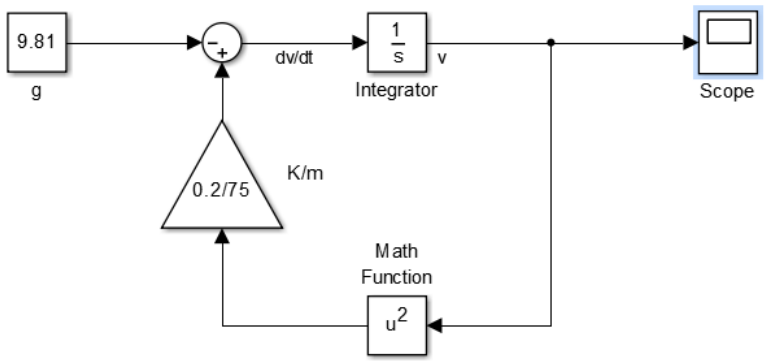
I *Function* velg *square.* Da blir det som kommer inn på inngangen opphøyd i annen. Trykk OK. Nå skal Math Function-blokken din se slik ut:



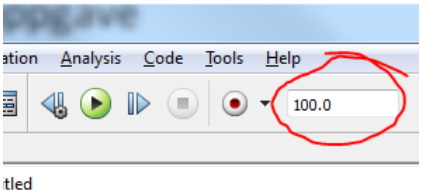
Nå kan vi sette sammen modellen vår slik:



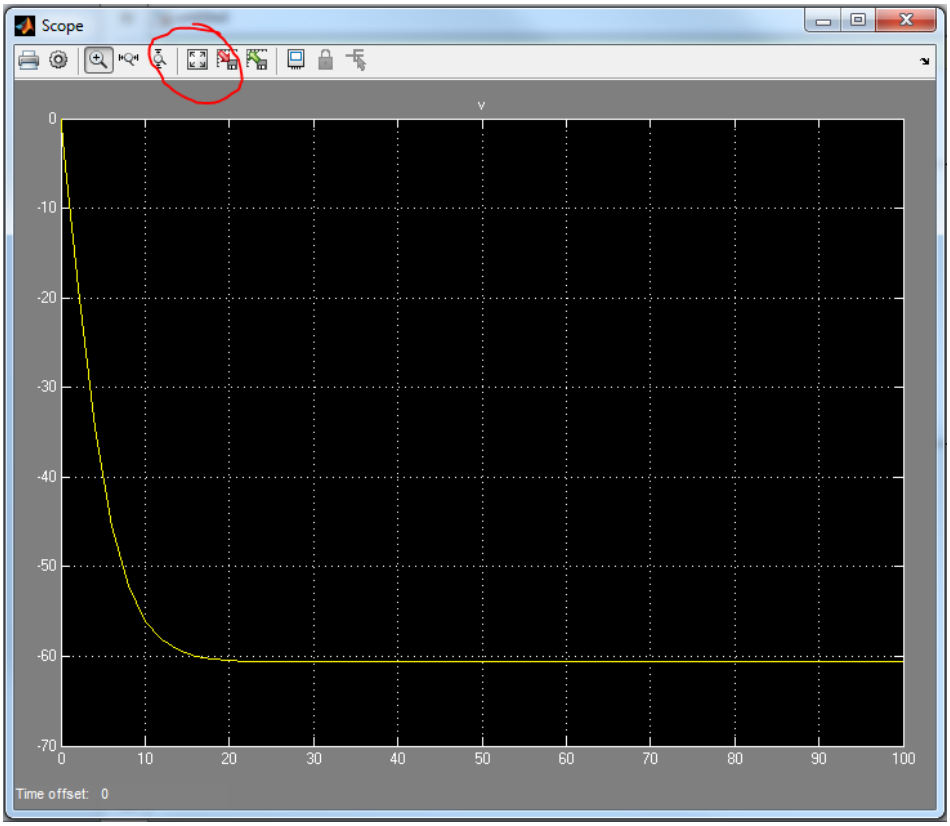
Nå har vi realisert differensiallikningen vår ($\frac{dv}{dt} = \frac{K}{m}v^2 - g$), og hastigheten *v* kan leses på utgangen av integratoren! Dobbeltklikk på integratoren og sjekk at *Initial* Condition er satt til 0. Dette betyr i vårt eksempel at hastigheten til Kari er 0 m/s når simuleringen begynner. For å se hvordan *v* endrer seg over tid bruker vi blokken Scope:



Det siste vi må gjøre er å velge hvor lenge vi vil simulere. 100 sekunder er passende, og det skriver du inn her:



Da er vi klare til å simulere! Trykk på den grønne play-knappen, og dobbeltklikk deretter på Scope. Da skal forhåpentligvis et vindu som ser slik ut åpnes:



Dersom ikke hele grafen vises, kan du prøve å trykke på *autoscale,* markert med en sirkel i bildet over. Her ser vi hvordan hastigheten til Kari endrer seg fra 0 m/s og stabiliserer seg på cirka -61 m/s (negativt fordi hun faller nedover), noe som tilsvarer nesten 220 km/t !