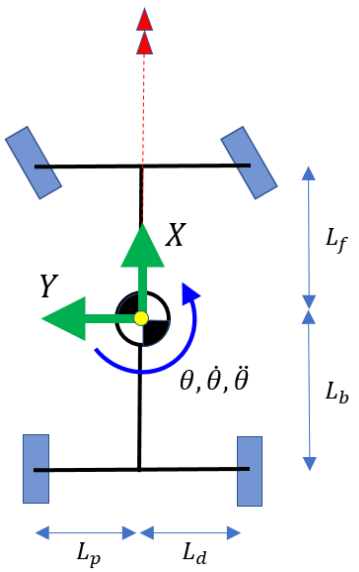
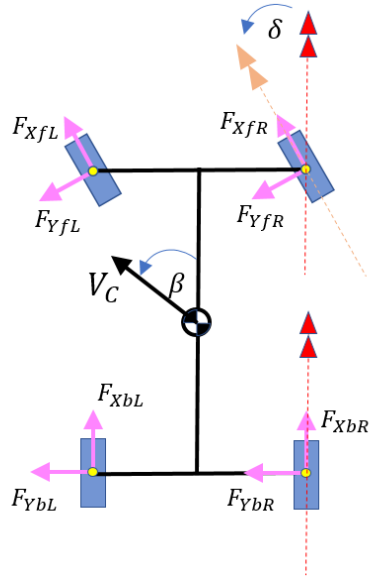
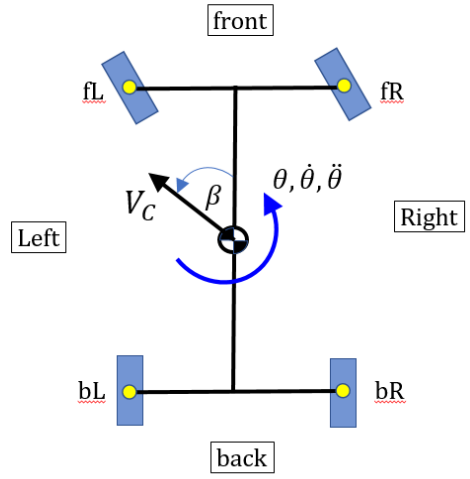
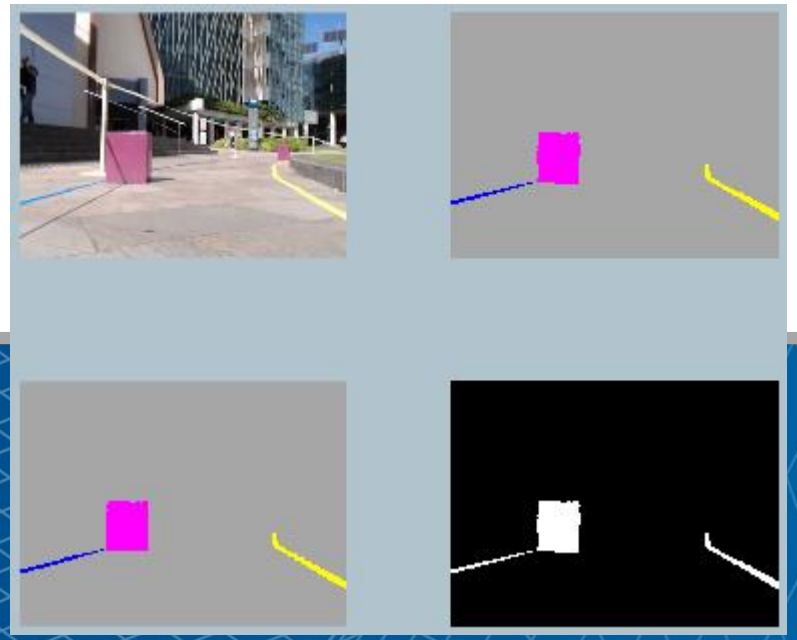
DEMO Guide and comments

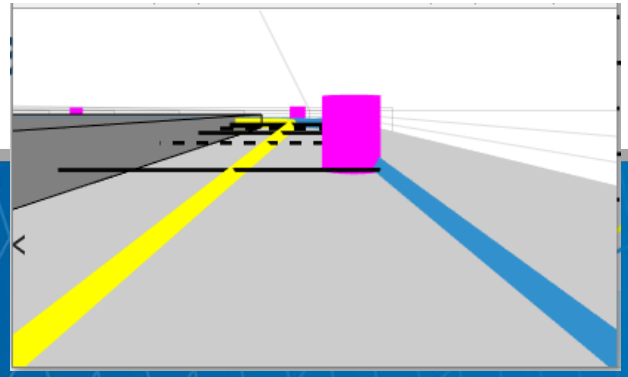
- *Droid Racing*

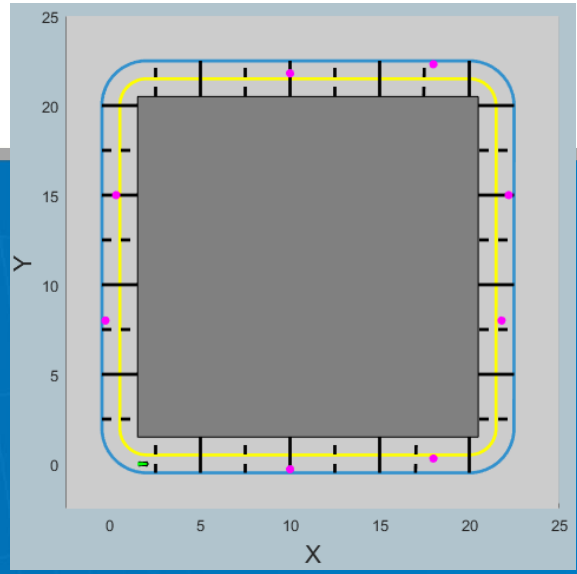












Brad Horton
Engineer
MathWorks

R2017b

Context

R2017b

NOTE: content developed
Using release R2017b

What you need to run the demo:

- R2017b (or NEWER)
  - MATLAB
  - Simulink
  - Stateflow
  - Symbolic Maths Toolbox
  - Image Processing Toolbox
  - Computer Vision System Toolbox
  - MATLAB Coder
  - Simulink Coder
  - Embedder Coder
  - Control System Toolbox
  - Simulink Control Design

These products will probably be on your University TAH campus license.

If they're not … then contact the **MathWorks**

OR

help yourself here:

https://www.mathworks.com/academia/student-competitions/droid-racing.html

The context of the demo:

R2017b

NOTE: content developed Using release R2017b

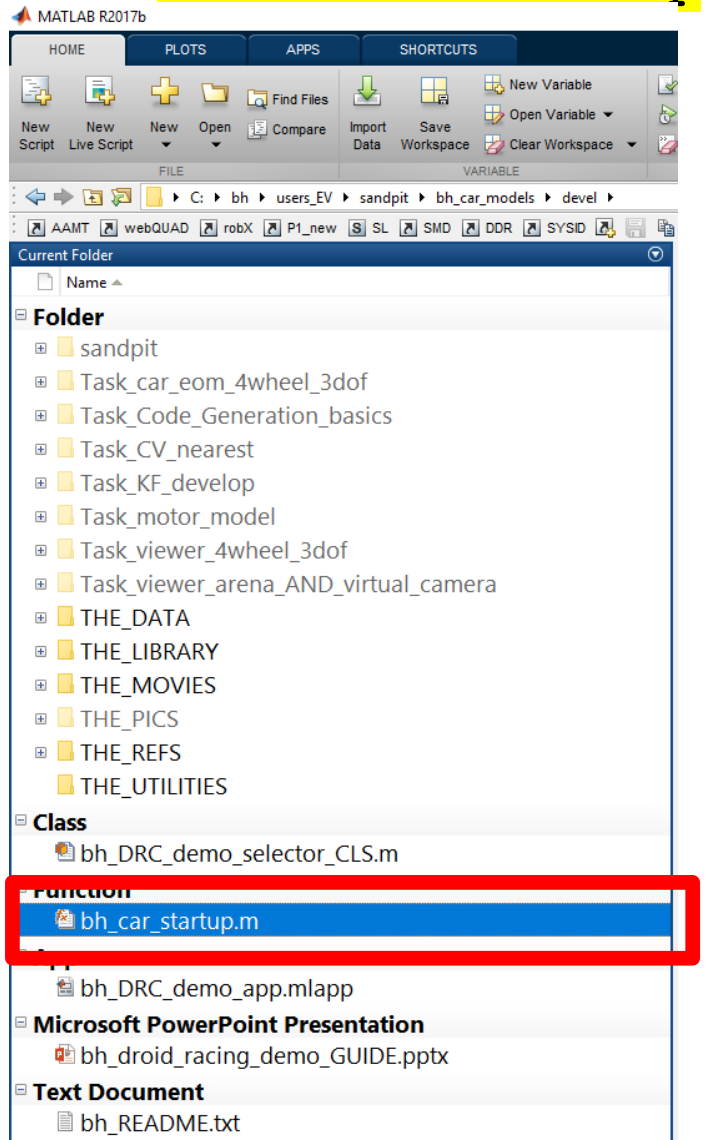
- Used for Edu engagements
  - Professors teaching machine dynamics
  - Professors teaching introductory image processing
  - The Droid Racing Challenge student competition

- Emphasise COMPUTATIONAL Thinking
  - Symbolic toolbox usage for equation derivation
    - Does the tedious stuff …. Brain is still needed for the physics/engineering stuff
  - Simulink for simulating models containing the derived equations of motion
  - MATLAB for prototyping a Computer Vision Algorithm
  - Automatic conversion of MATLAB algorithm into standalone C/C++ code.
  - Simulink for simulating a complete SYSTEM
    - car dynamics and Computer vision

Launch the demo:

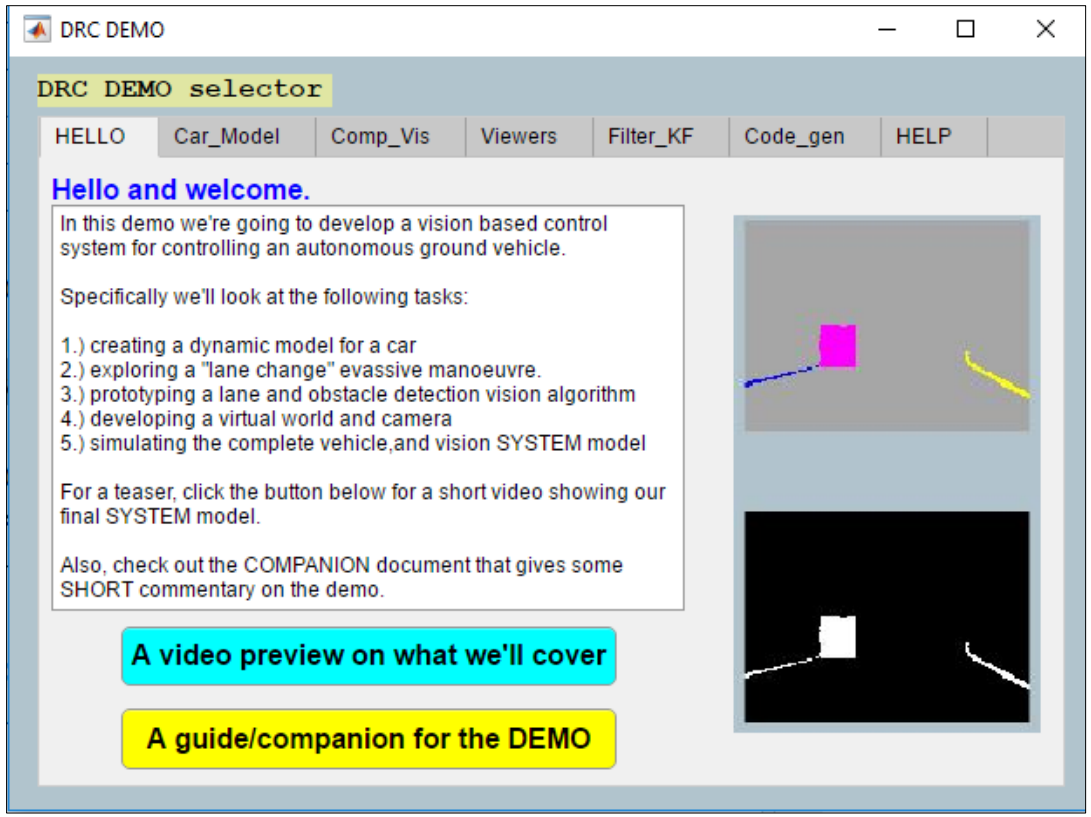
- Run the start-up function

```
>> bh car startup
```

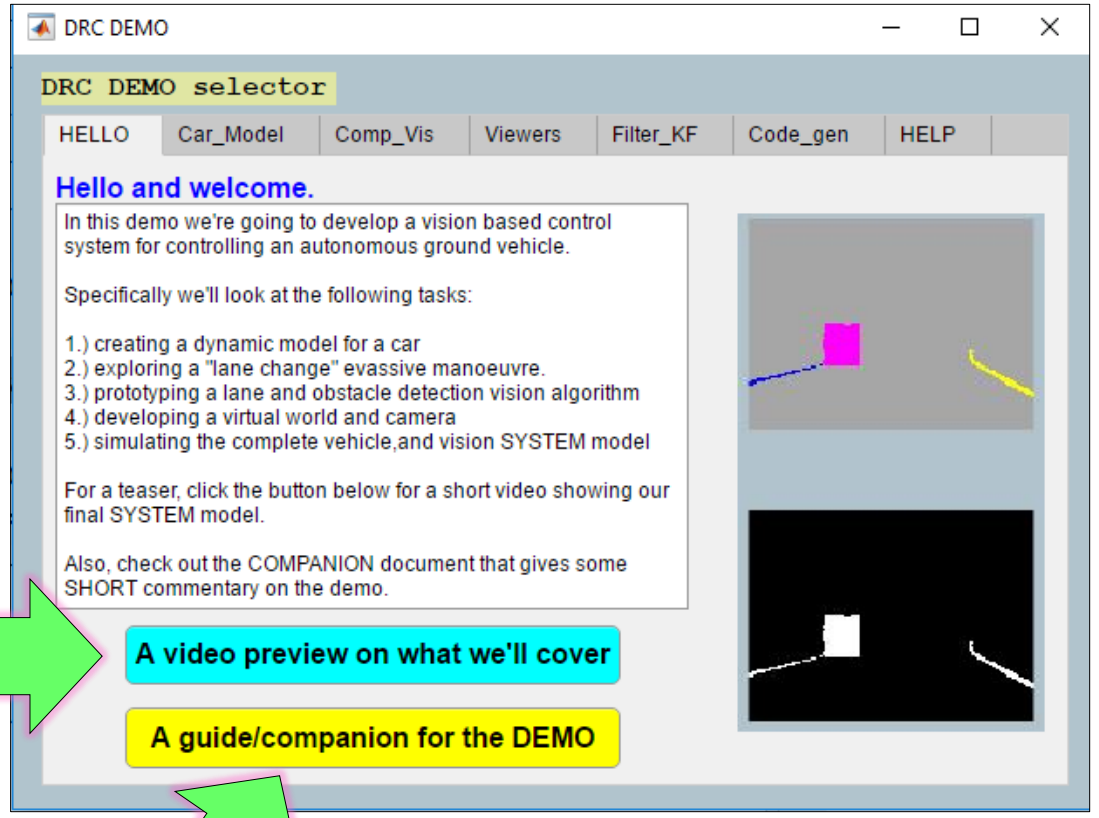


1.

After running the start-up function
the DEMO navigator APP
will appear:



Hello and welcome:

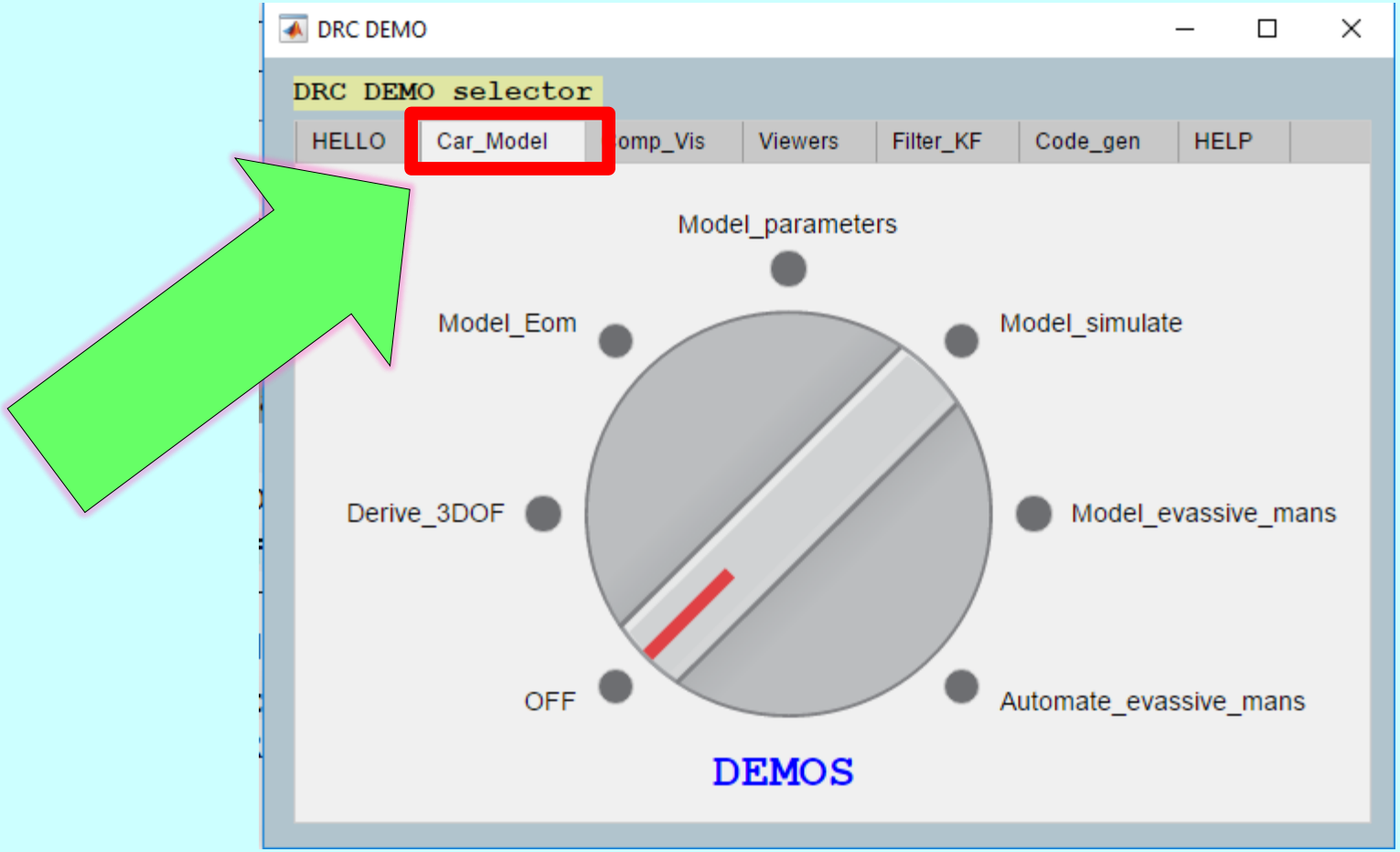


A 2 minute video showing the "final" system model that incorporates our car dynamics AND our custom vision algorithm.

ALL of the mini demos that we cover will lead us to this final SYSTEM level simulation.

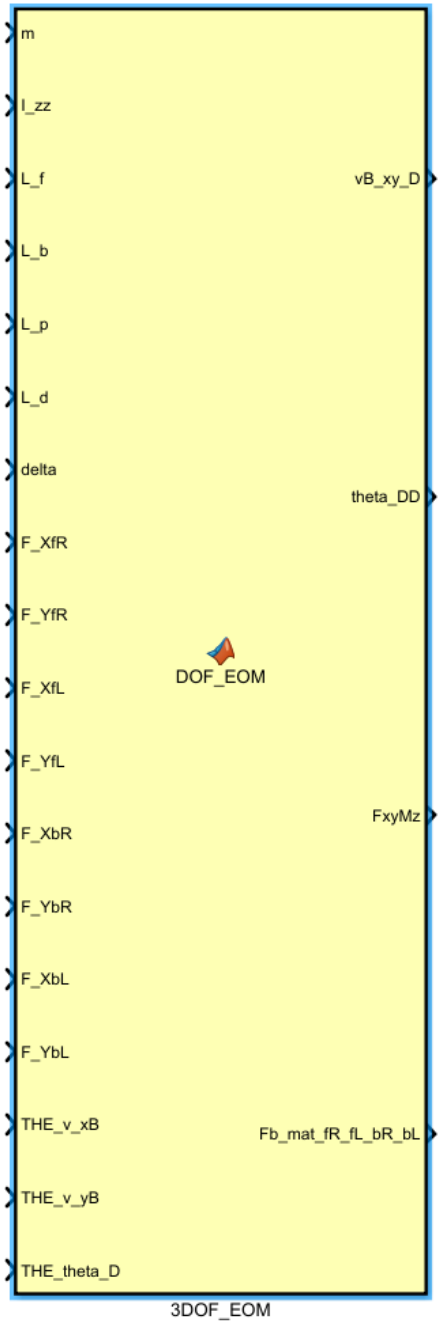
Opens a companion document (either pptx OR pdf) that gives some short comments on each demo.

The Car Model.



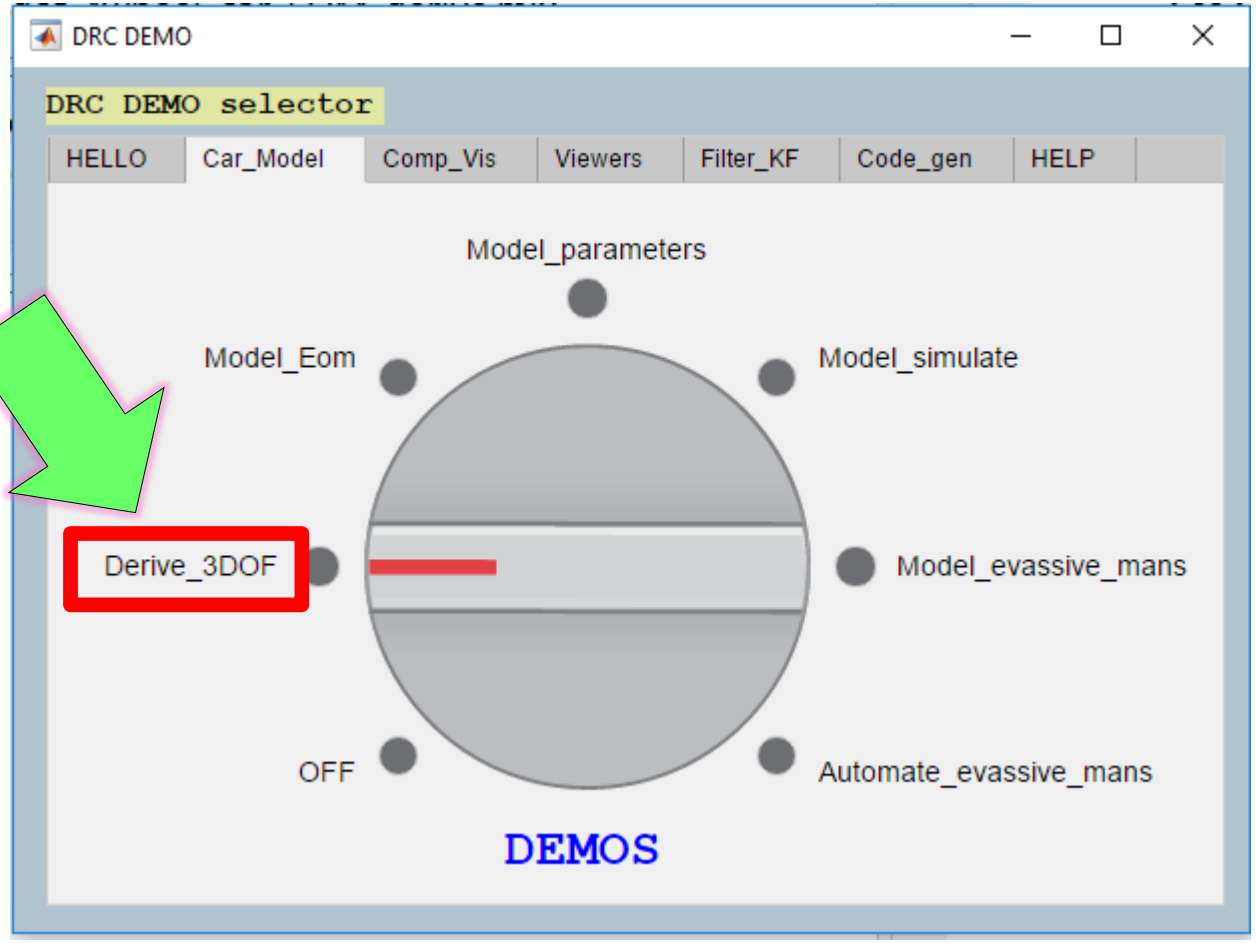
Car Model: overview

- Derive equations of motion of a 3-dof car model
  - Newton's 2nd Law
  - Use a simple rolling resistance model for LONGITUDINAL friction
  - Use a simple linear sideslip model for LATERAL friction
- Use Symbolic toolbox to
  - Derive equations of motion
  - Convert derived equations into a "**YELLOW**" block for Simulink
- Simulate the model in Simulink
  - Explore a lane change evasive manoeuvre
    - Will be our go to strategy for avoiding obstacles



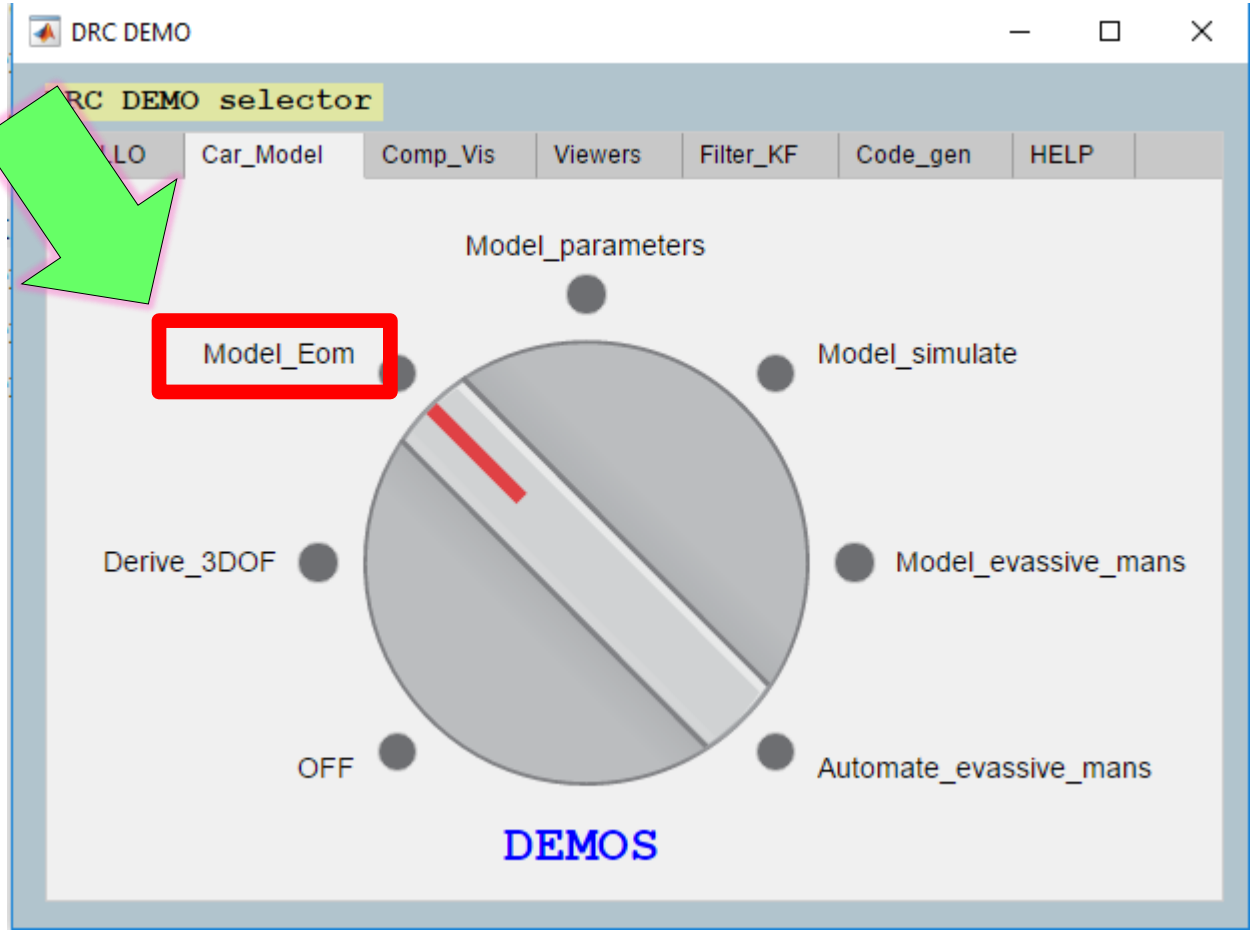
Comments on the **Car model** demo: `Part 1 of 6`

- Opens a LIVE script in MATLAB
- **YOU** manually run this script, to derive the Equations of motion for the 3-dof car model.
- Explore the derivation, and NOTE:
  - The use of the Symbolic Math toolbox
  - The automatic conversion of the car's ODEs into a Simulink block
    - The little **"YELLOW"** block



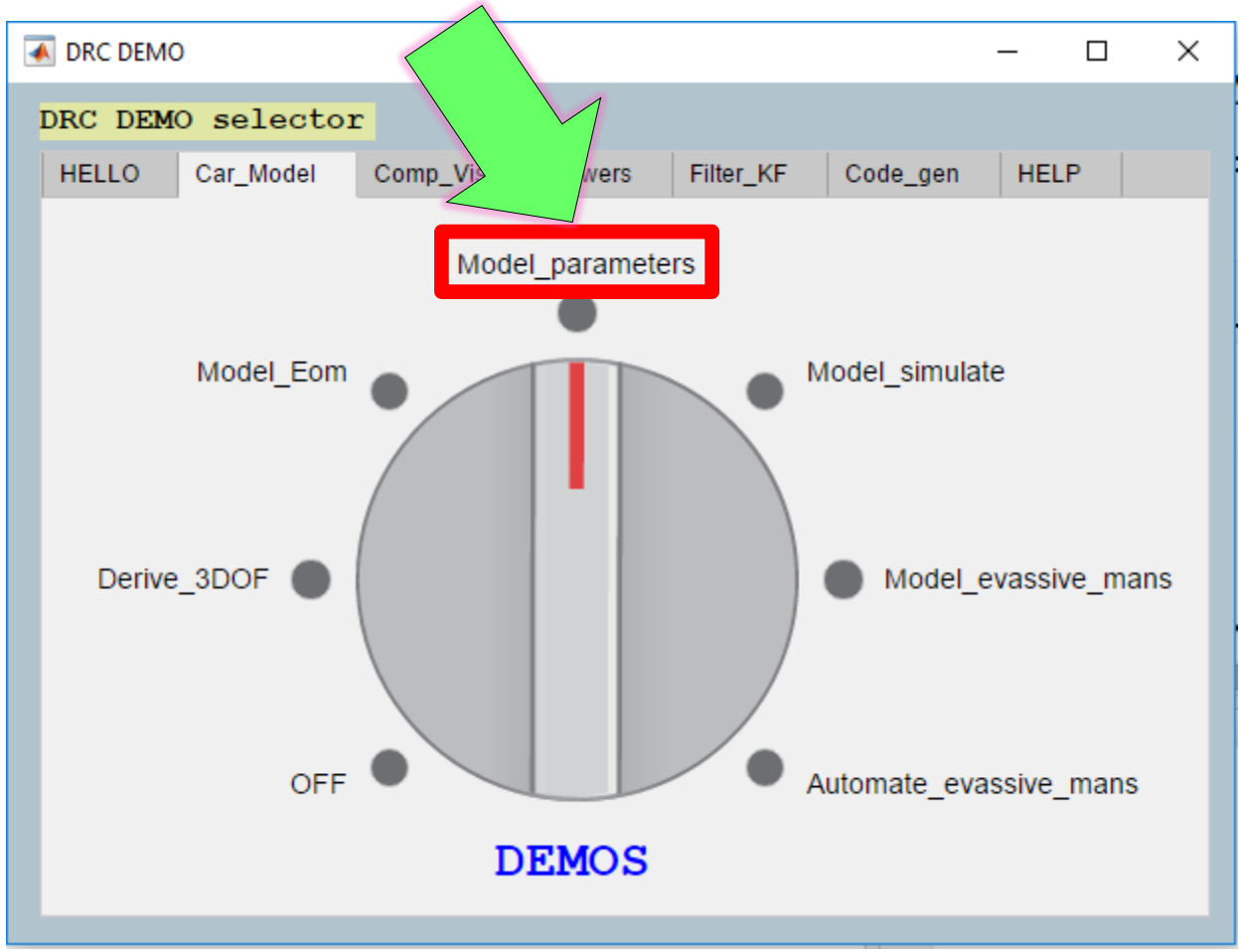
Comments on the **Car model** demo: `Part 2 of 6`

- Opens a Simulink component model that contains our derived equations of motion for the car dynamics.
- You **Do NOT run** this model.
- This model is referenced by other models
- **So what do you do with it ?**
  - Just have a look inside it.
  - Note the YELLOW block that contains our derived equations of motion.
  - Note how we integrate accelerations to get velocities …. Etc.



Comments on the **Car model** demo: `Part 3 of 6`

- Opens a LIVE script in MATLAB
- You **Do NOT run** this script.
- The script defines properties for our car model (eg: mass, geometry, friction co-efficients , etc)
- **So what do you do with it ?**
  - Just be aware that this script acts as a “Data Dictionary” for characterising our car model.
  - In later examples, where we actually simulate models, this “car parameter” script is run behind the scenes to define variables into the MATLAB workspace.

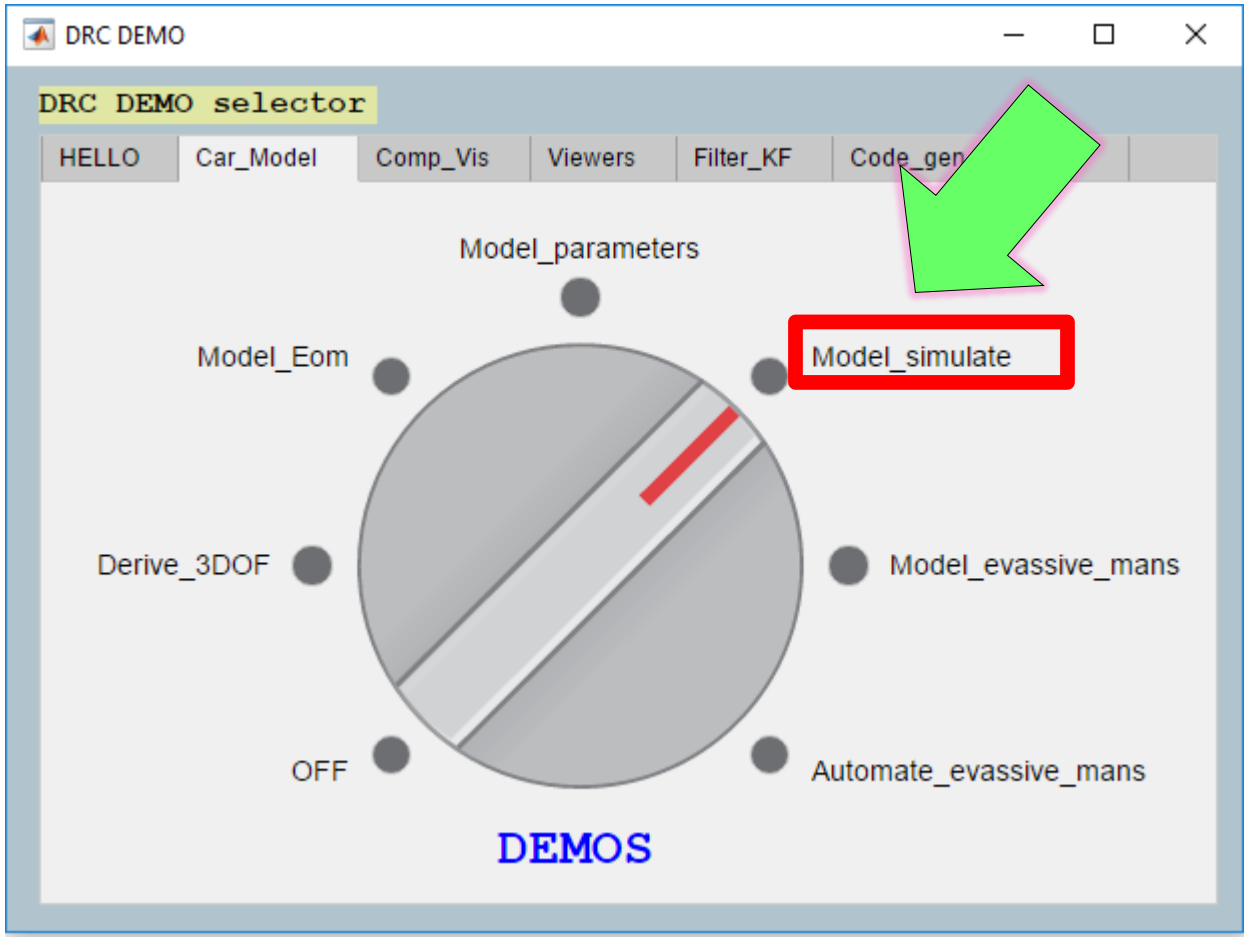


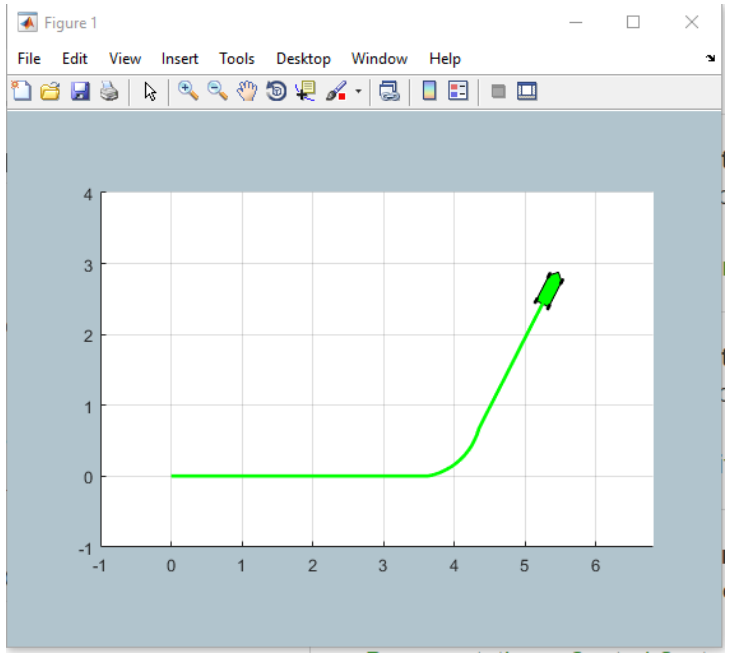
Comments on the **Car model** demo: `Part 4 of 6`

- Opens a Simulink model
  - Contains our derived 3-dof car dynamics

- **So what do you do with it ?**
  - **YOU** manually run the model.

  - Observe the car behaviour when we apply a finite duration pulse to the steering FRONT wheels of the car





Comments on the **Car model** demo: `Part 5 of 6`

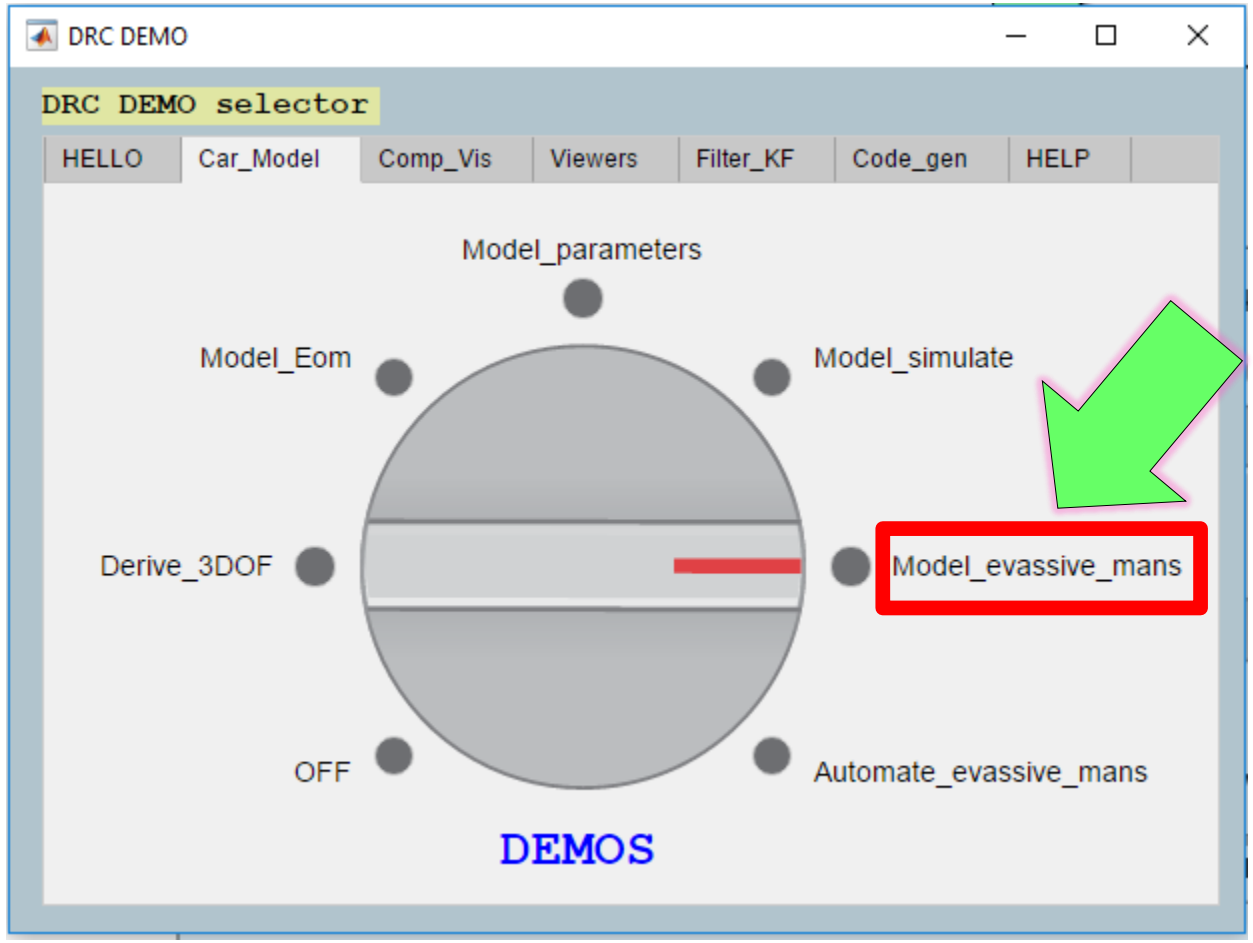
- Opens a Simulink model
  - Contains our derived 3-dof car dynamics

- In this model we've included:
  - a speed control system that makes the car move at a specified speed.

  - We have also added a simple FRONT wheel steering angle profile to show how the car can perform a "lane changing" manoeuvre.
    - We'll use this as a core component for designing an OBSTACLE avoidance system

- **So what do you do with it ?**
  - **YOU** manually run the model.

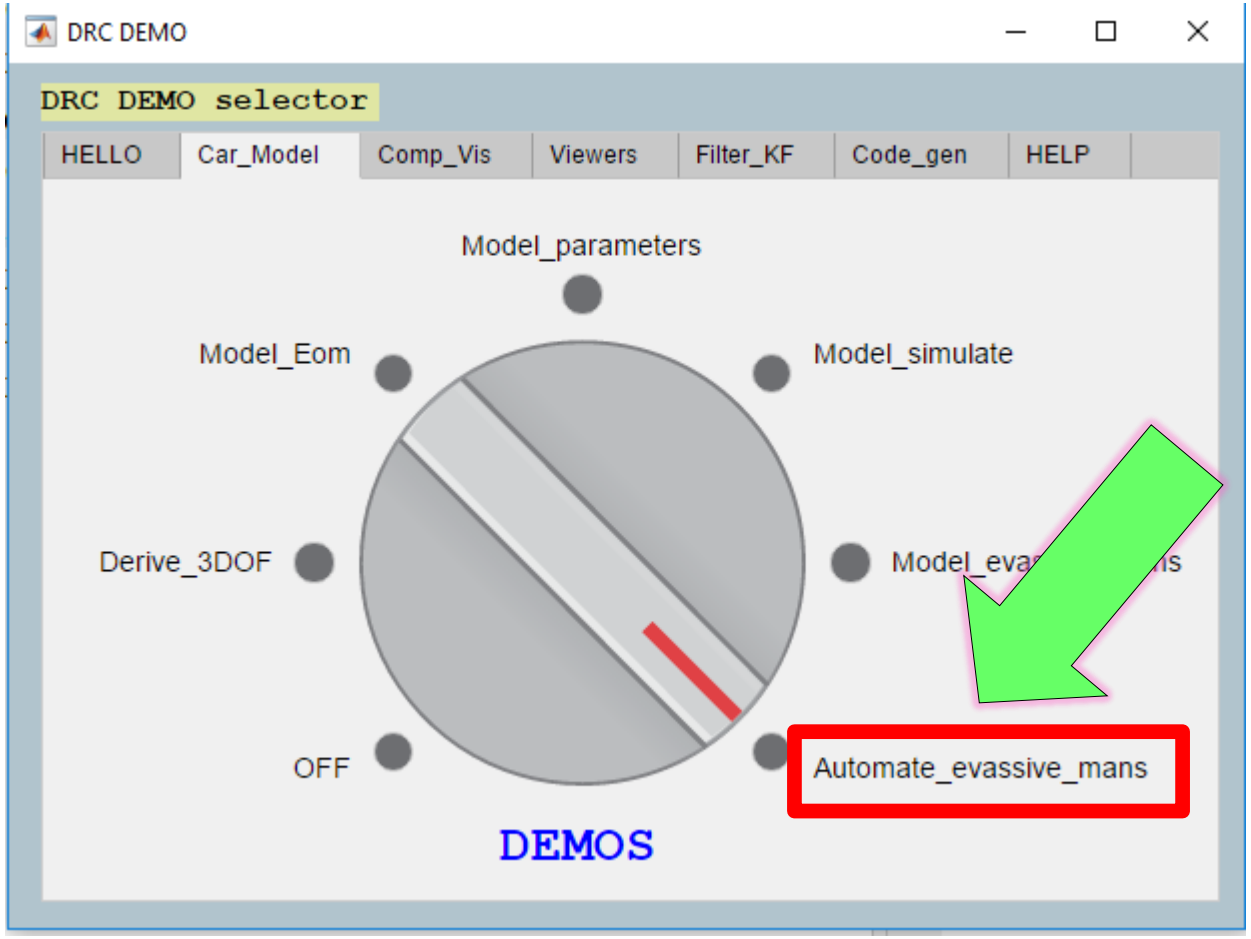
  - Interact with the DIALS embedded in the model and RERUN the model.
    - observe the impact that these parameters have on the car's behaviour.



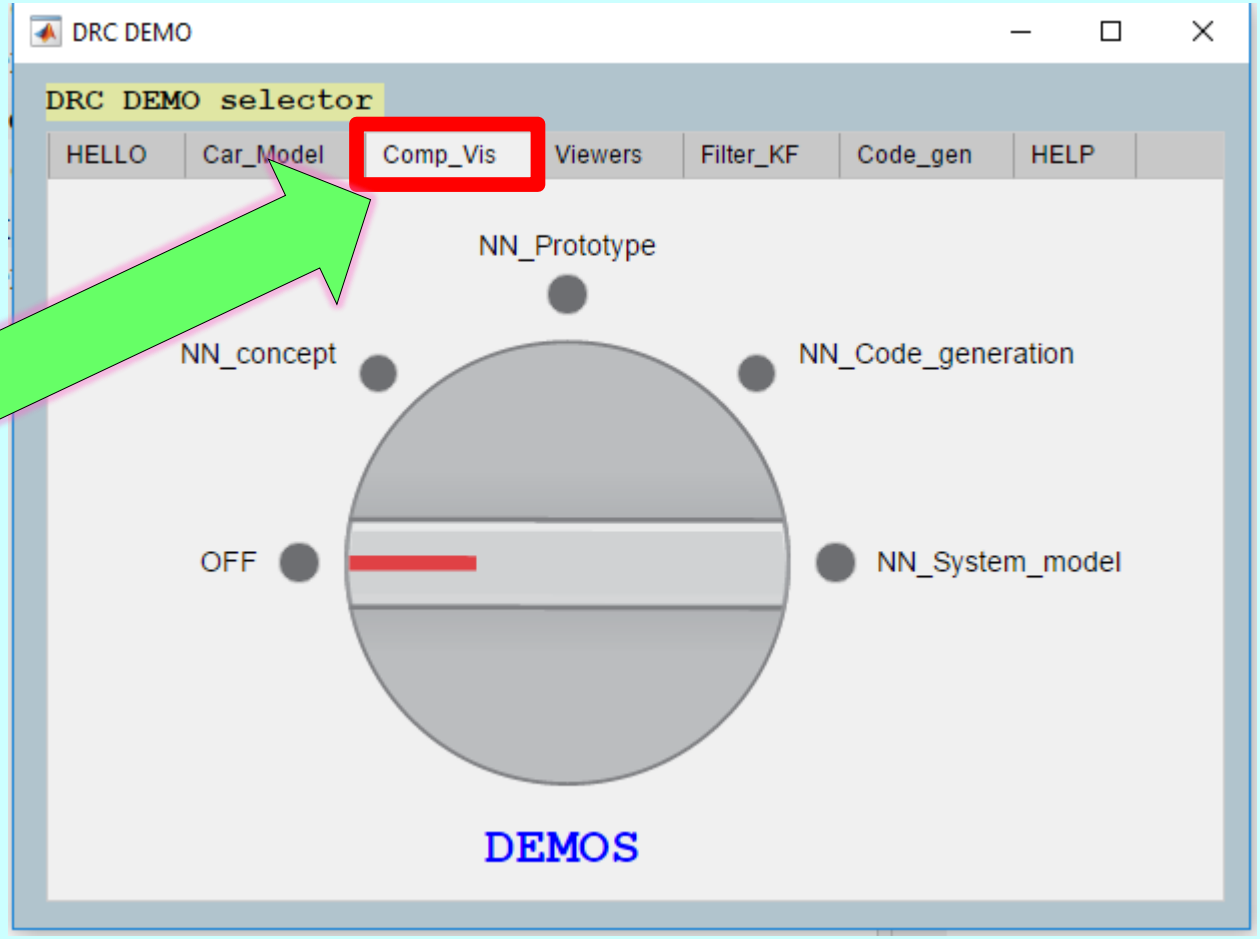
Comments on the **Car model** demo: `Part 6 of 6`

- Opens a LIVE script in MATLAB

- **So what do you do with it ?**
  - **YOU** manually run this script.
  - It shows how to automate the “calling “ of a Simulink model from MATLAB.
  - We use this technique to explore the impact of certain parameters that characterise our lane changing steering manoeuvre



The Vision algorithm.

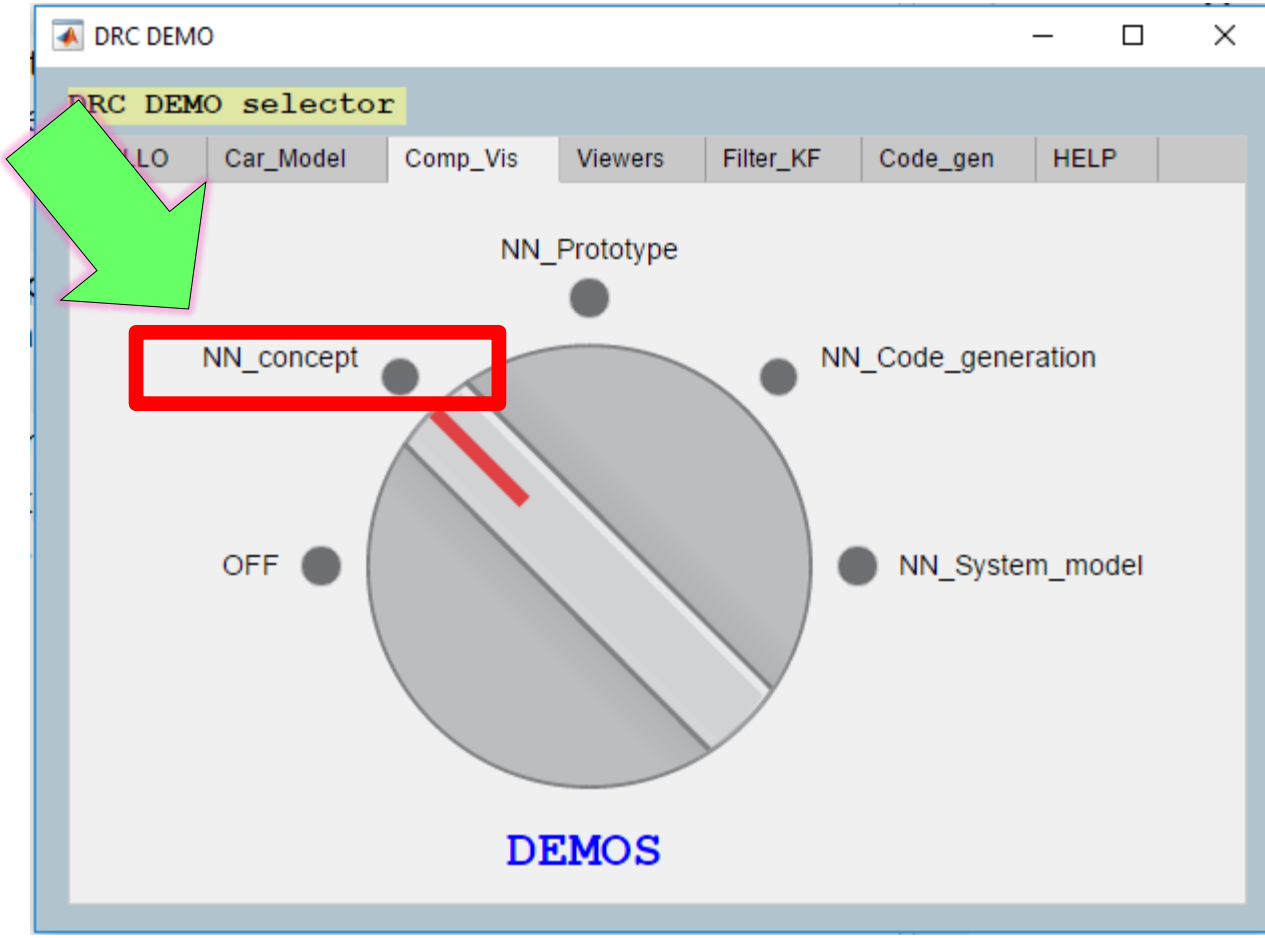


The vision algorithm: overview

- Explore a Nearest Neighbour (NN) approach for identifying
  - BLUE and YELLOW lanes
  - PURPLE obstacles
- Implement the NN concept into a MATLAB function
  - Test the algorithm with real world video footage.
- Simulate the car and vision components in a SYSTEM simulation
  - So we'll use our CAR dynamics AND our Vision algorithm … ALL in the same model

Comments on the **Vision algorithm** demo: Part 1 of 4

- Opens a LIVE script in MATLAB
- **YOU** manually run this script, explore a Nearest Neighbour (NN) concept to identifying target “things” within an image.
- **So what do you do with it ?**
  - **YOU** manually run the script.
  - Explore the algorithm
    - In the next demo we’ll implement the NN algorithm into a reusable MATLAB function …. AND test it on real world video footage.



Comments on the **Vision algorithm** demo: Part 2 of 4

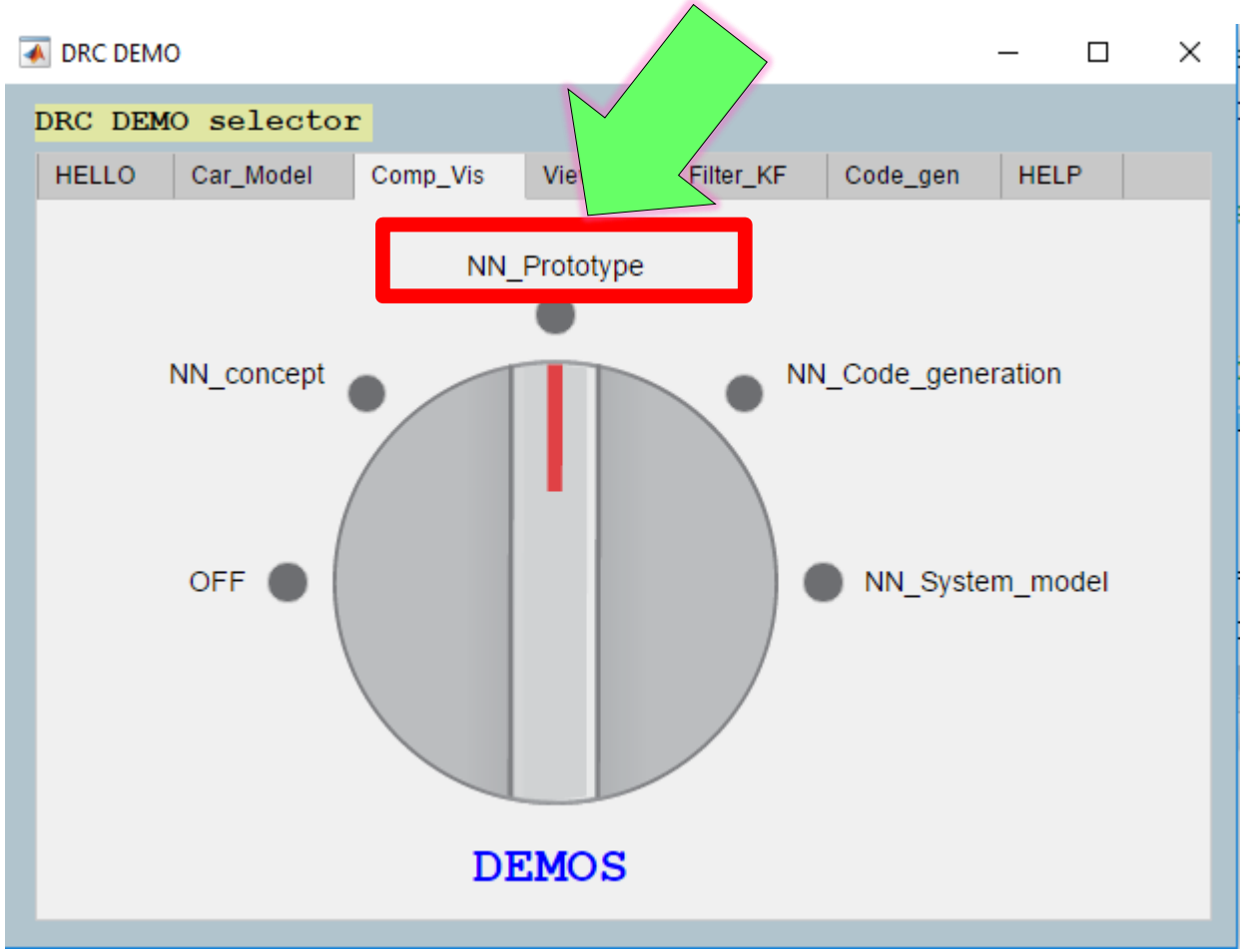
- Opens a collection of MATLAB files
  - A main script
    `bh_test_kf_video_for_CODEGEN.m`
  - 3 function *.m files

- The 3 functions define our algorithm

- The script is a TEST harness for testing our algorithm with a recorded video file.

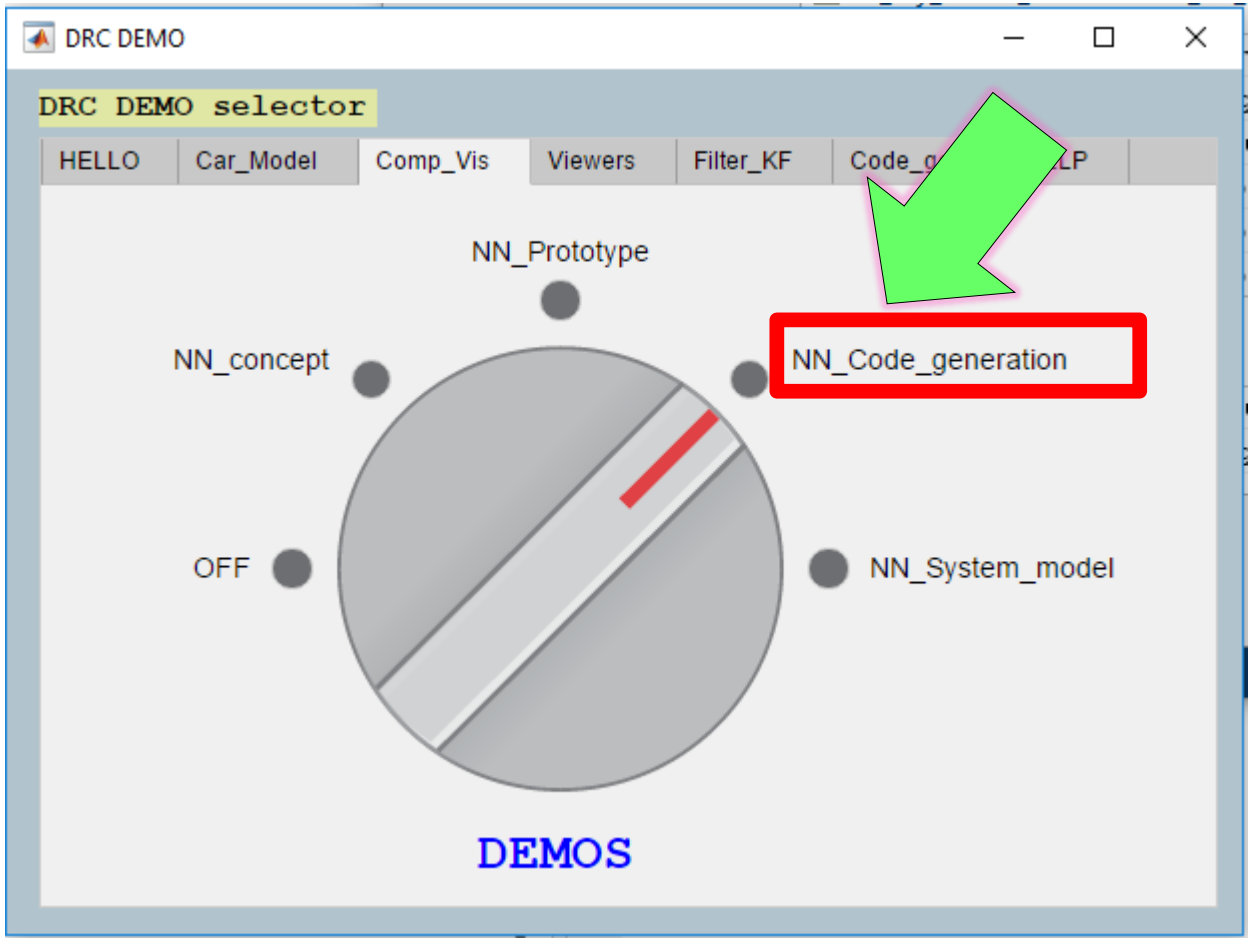
- **So what do you do with it ?**
  - **YOU** manually run the TEST script.

  - NOTE how the algorithm identifies:
    - BLUE and YELLOW lanes
    - PURPLE obstacles



Comments on the **Vision algorithm** demo: Part 3 of 4

- Opens the MATLAB Coder APP
  - An existing project is opened
- The project shows how to automatically convert our vision algorithm defined by our 3 MATLAB functions, into standalone C-code
- **So what do you do with it ?**
  - Just browse through the project.
  - ***ATTENTION:*** *We'll take a closer look at the code generation task when we do the "Code_gen" demo TAB.*



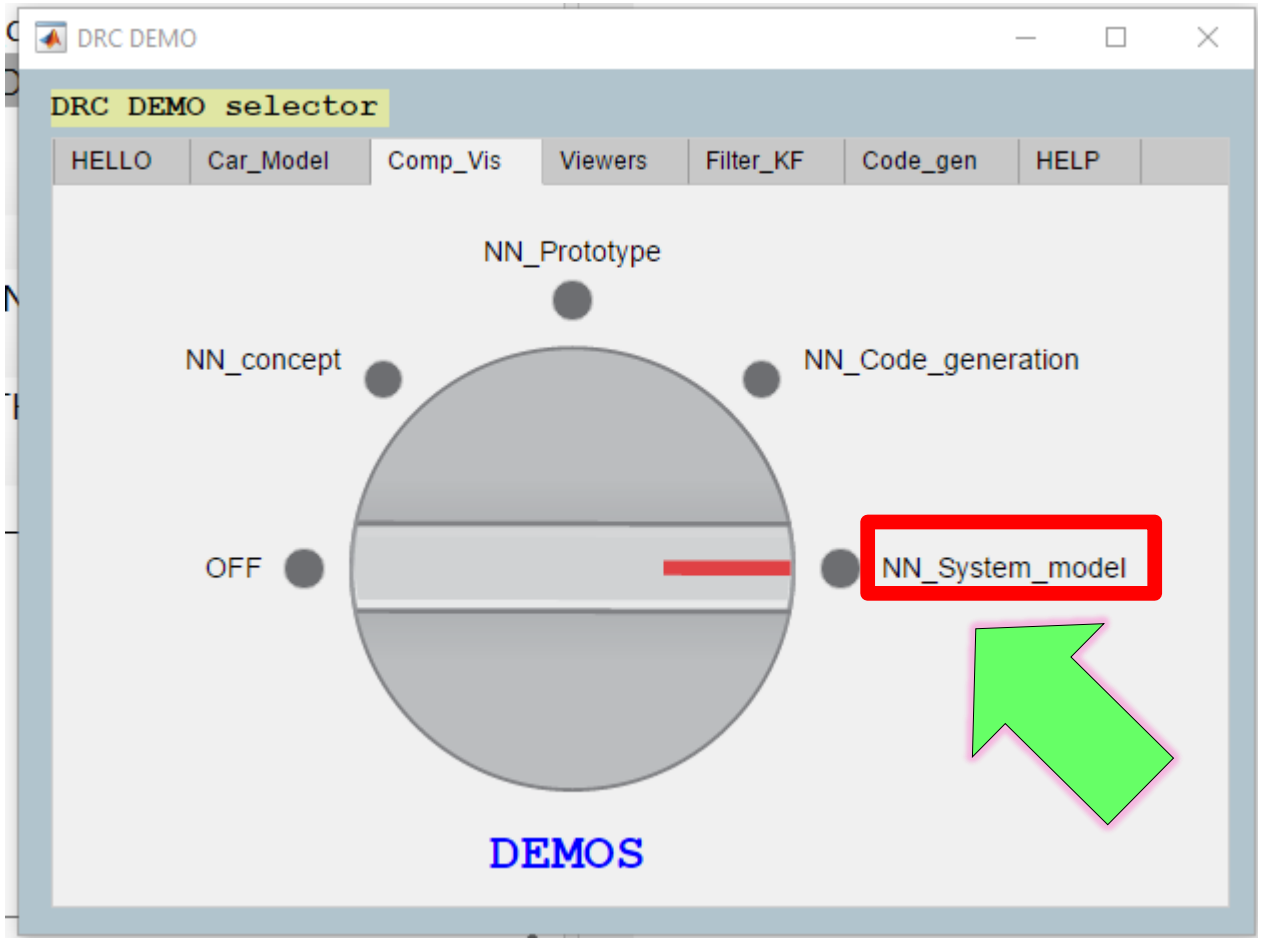
Comments on the **Vision algorithm** demo: Part 4 of 4

- Opens a Simulink model

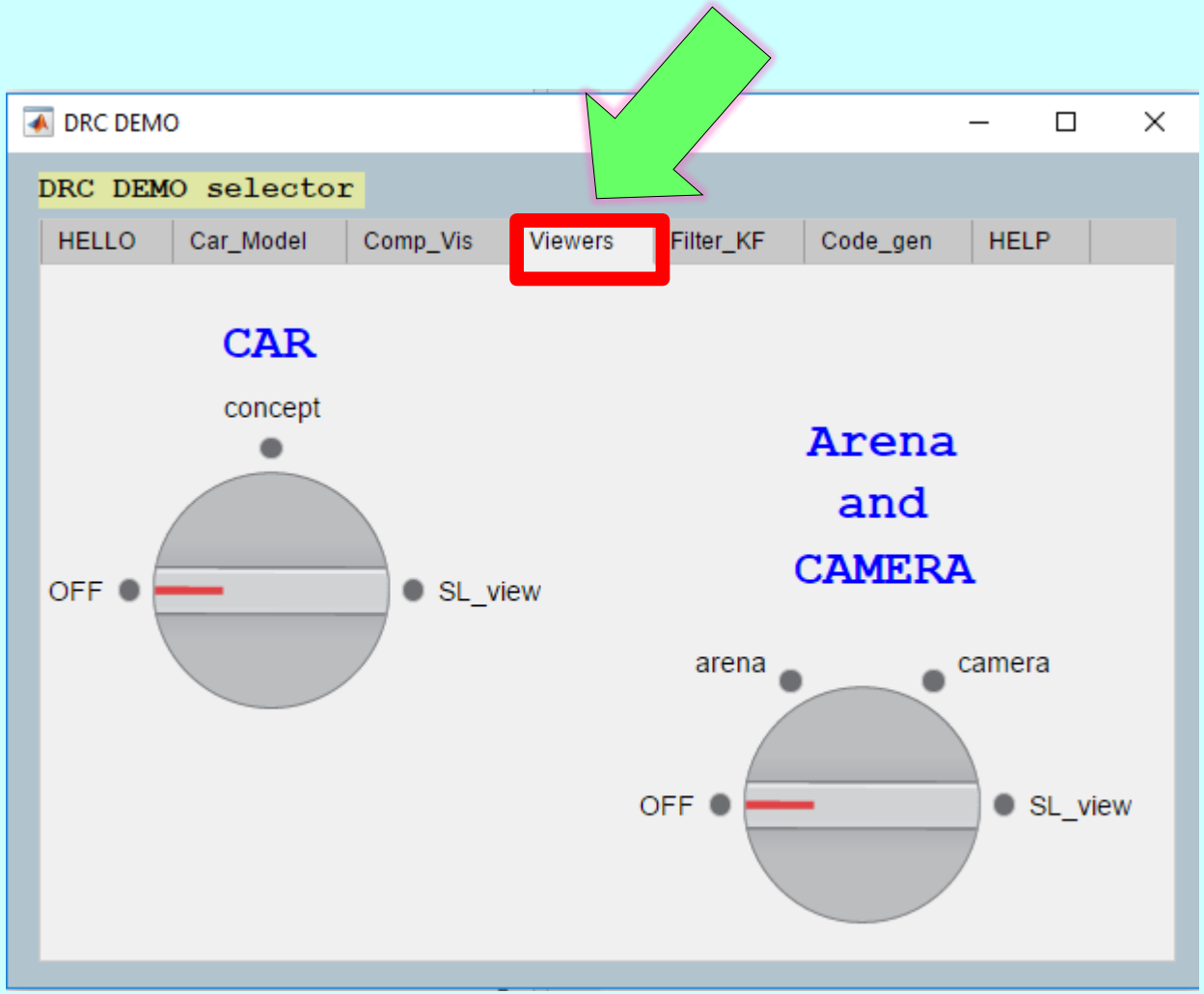
- The model contains:
  - Our car dynamics
  - Our Vision algorithm
  - A VIRTUAL arena (or race track)
  - A VIRTUAL camera

- **So what do you do with it ?**
  - **YOU** manually run the model.

  - Observe how our vision algorithm is able to control the car.
    - the car avoids PURPLE obstacles.
    - the car stays inside the BLUE and YELLOW lanes



The VIEWERS.


DRC DEMO
DRC DEMO selector
HELLO
Car_Model
Comp_Vis
Viewers
Filter_KF
Code_gen
HELP
CAR
concept
OFF
SL_view
Arena
and
CAMERA
arena
camera
OFF
SL_view

The VIEWERS: overview

- Develop simple Visualization tools in MATLAB
  - A car viewer
  - A race track arena
  - A virtual camera
- Incorporate viewers into Simulink models
  - Use M-file S-functions
  - Simple wrappers that allow us to REUSE our MATLAB code within Simulink

Comments on the **VIEWERS** demo: Part 1 of 5

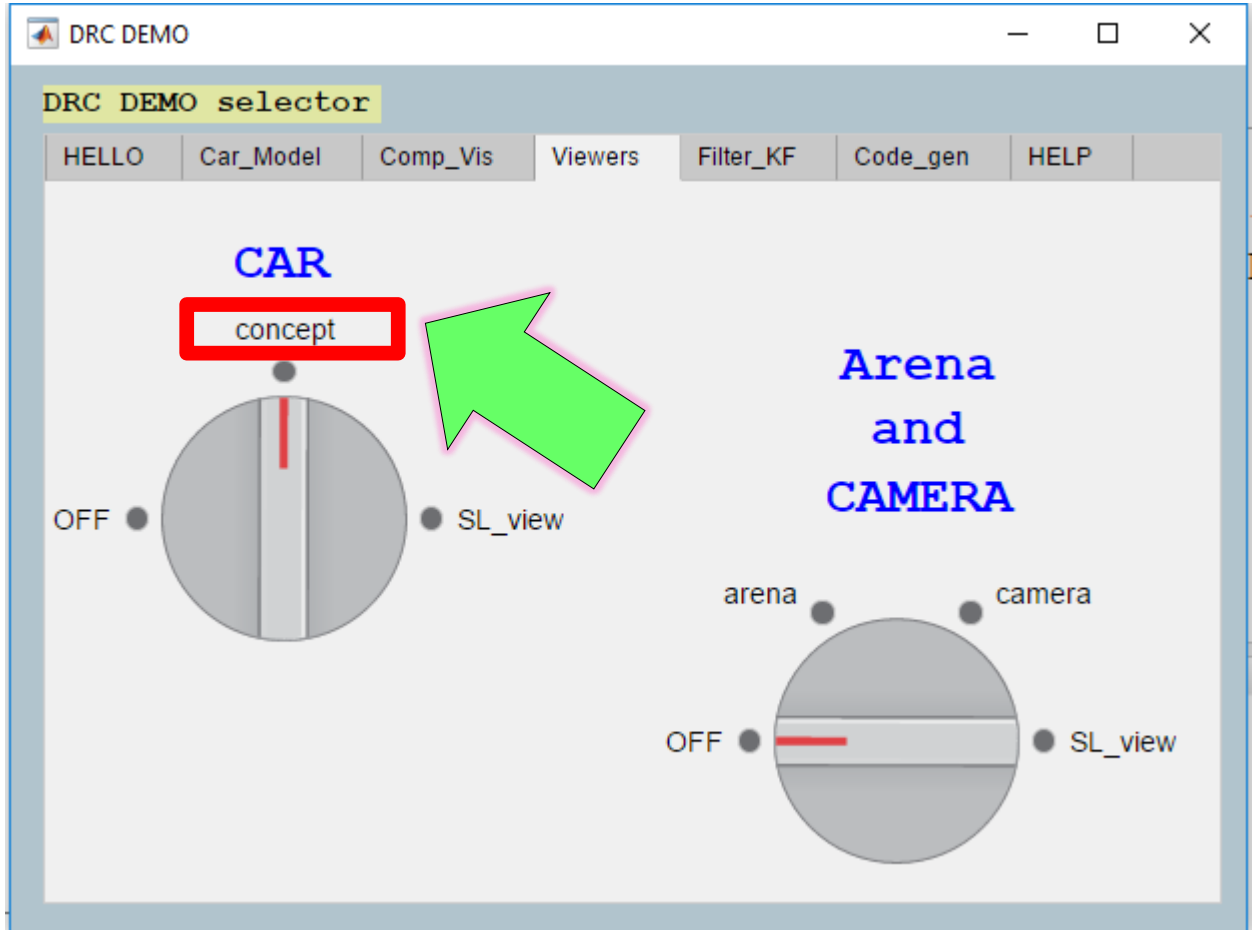
- Opens a collection of MATLAB files
  - A main script
    `bh_test_bh_3dof_4wheel_vehicle_viewer.m`
  - 1 MATLAB class *.m file

- The class *.m file creates a simple visualization of a car
  - Plots the pose (x, y, theta) of a car graphic

- The script is a TEST harness for testing the code defined by the MATLAB class.

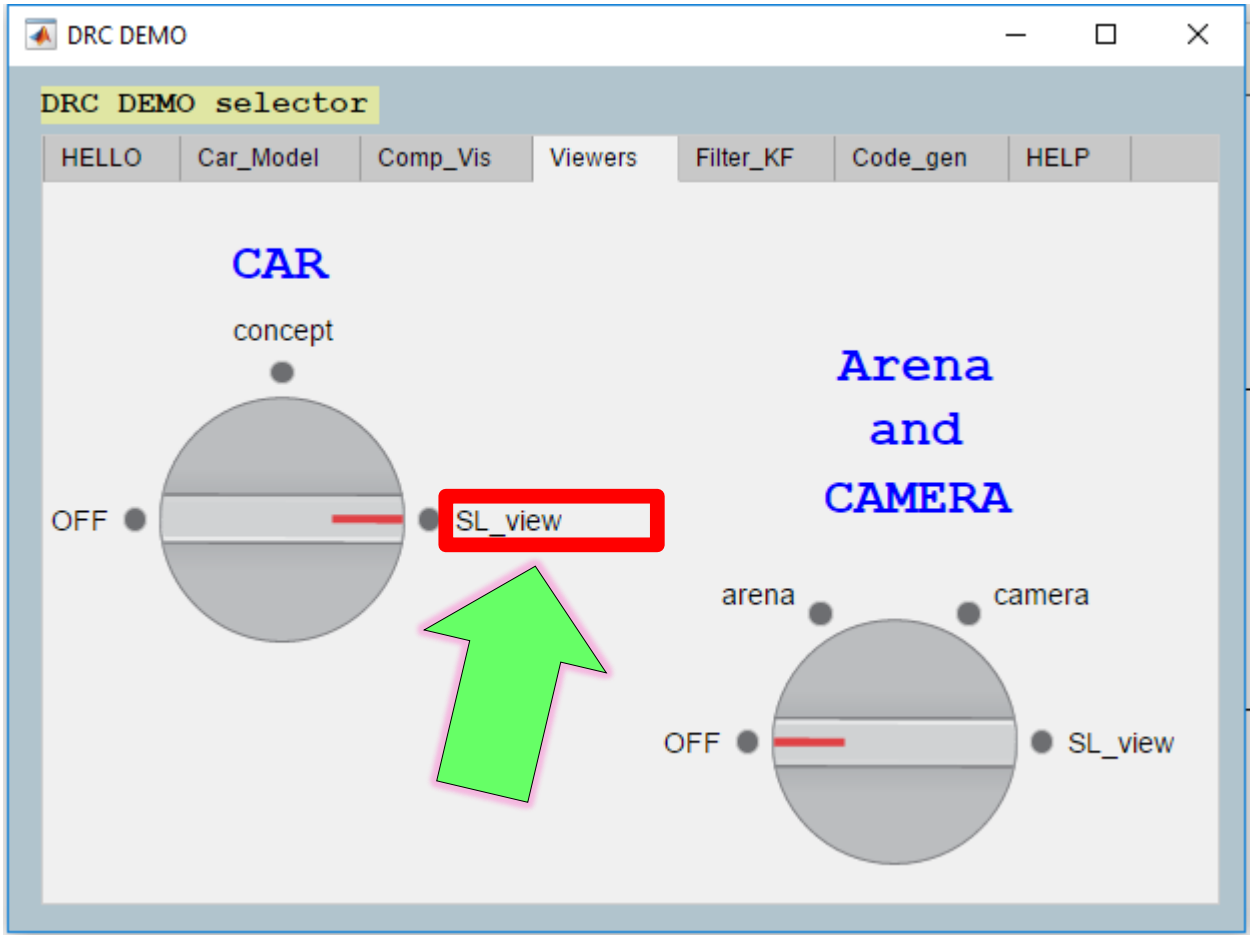
- **So what do you do with it ?**
  - **YOU** manually run the TEST script.

  - A very simple viewer



Comments on the **VIEWERS** demo: Part 2 of 5

- Opens a Simulink model
- The Simulink model contains a block that calls our car viewer code
- **So what do you do with it ?**
  - **YOU** manually run the model.
  - A very simple viewer



Comments on the **VIEWERS** demo: Part 3 of 5

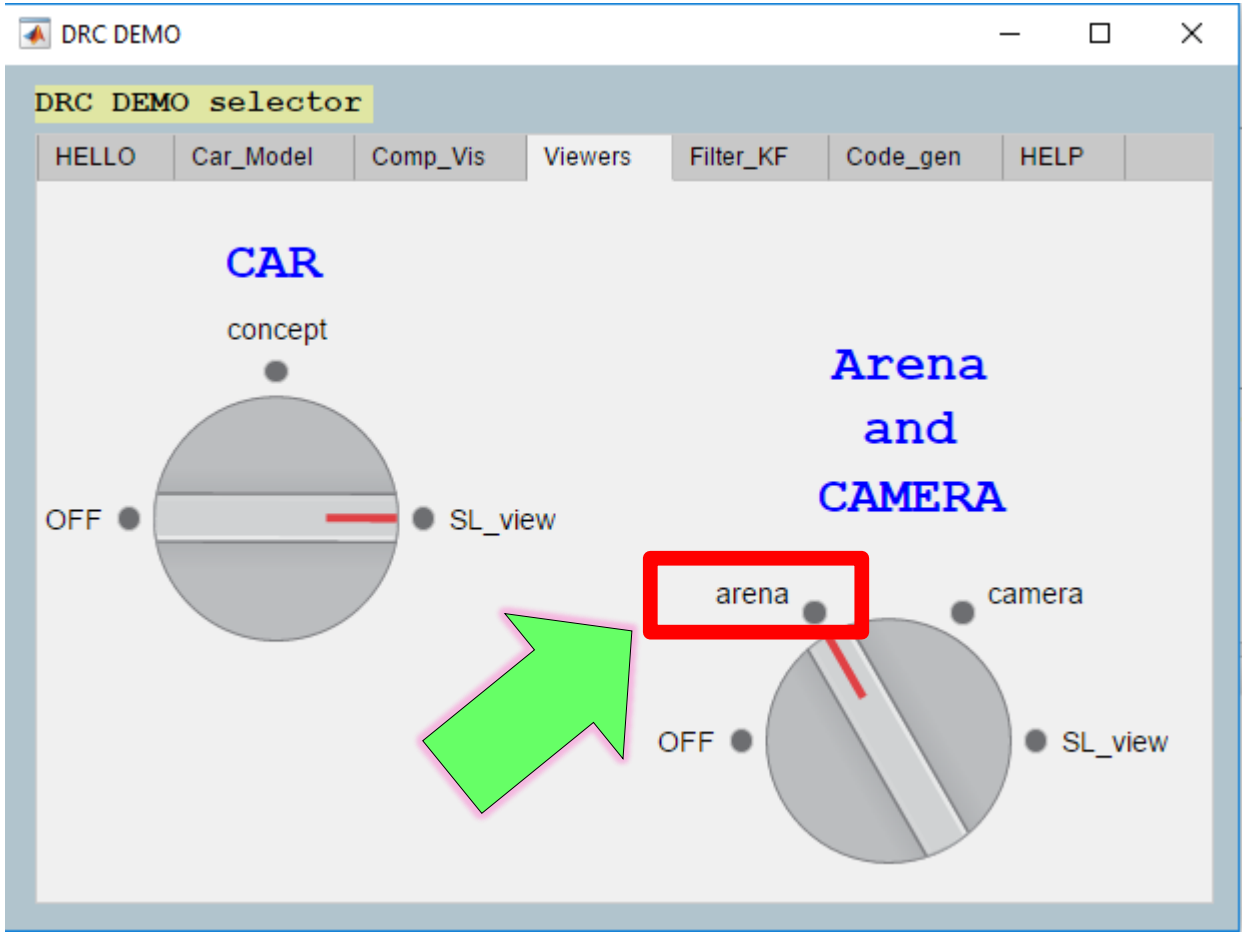
- Opens a collection of MATLAB files
  - A main script **`bh_test_arena.m`**
  - 1 Matlab class *.m file

- The class **`bh_3dof_4wheel_vehicle_arena_CLS.m`** file creates a simple visualization of a race track
  - BLUE and YELLOW lanes
  - PURPLE obstacles

- The script is a TEST harness for testing the code defined by the MATLAB class.

- **So what do you do with it ?**
  - **YOU** manually run the TEST script.

  - A very simple viewer



Comments on the **VIEWERS** demo: Part 4 of 5

- Opens a collection of MATLAB files
  - A main script **`bh_test_arena_and_cam.m`**
  - 2 Matlab class *.m file

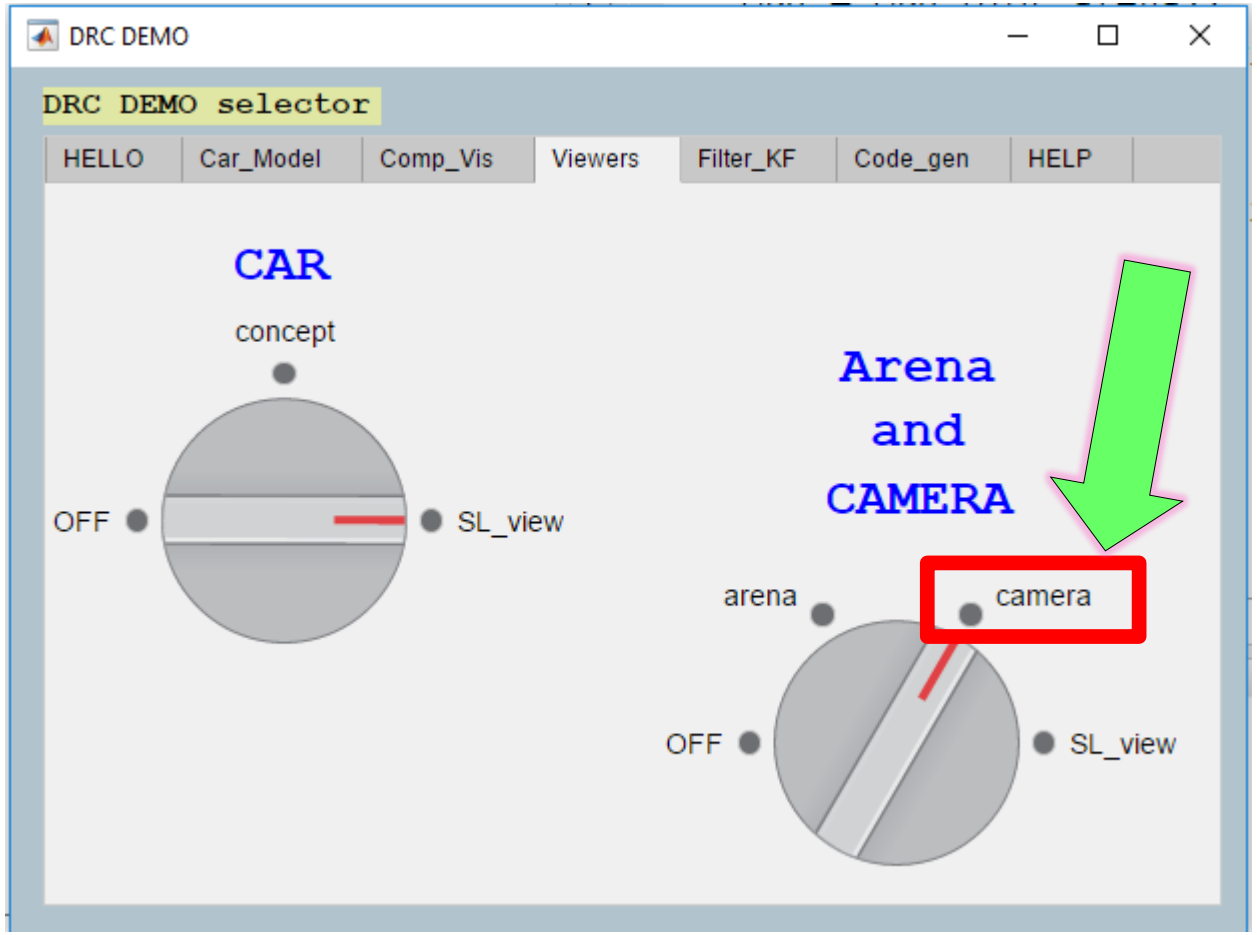
- The class **`bh_3dof_4wheel_vehicle_arena_CLS.m`** file creates a simple visualization of a race track
  - BLUE and YELLOW lanes
  - PURPLE obstacles

- The class **`bh_virtual_car_camera_CLS.m`** file creates a VIRTUAL camera
  - We can insert the camera into our VIRTUAL race track

- The script is a TEST harness for testing the code defined by the MATLAB class.

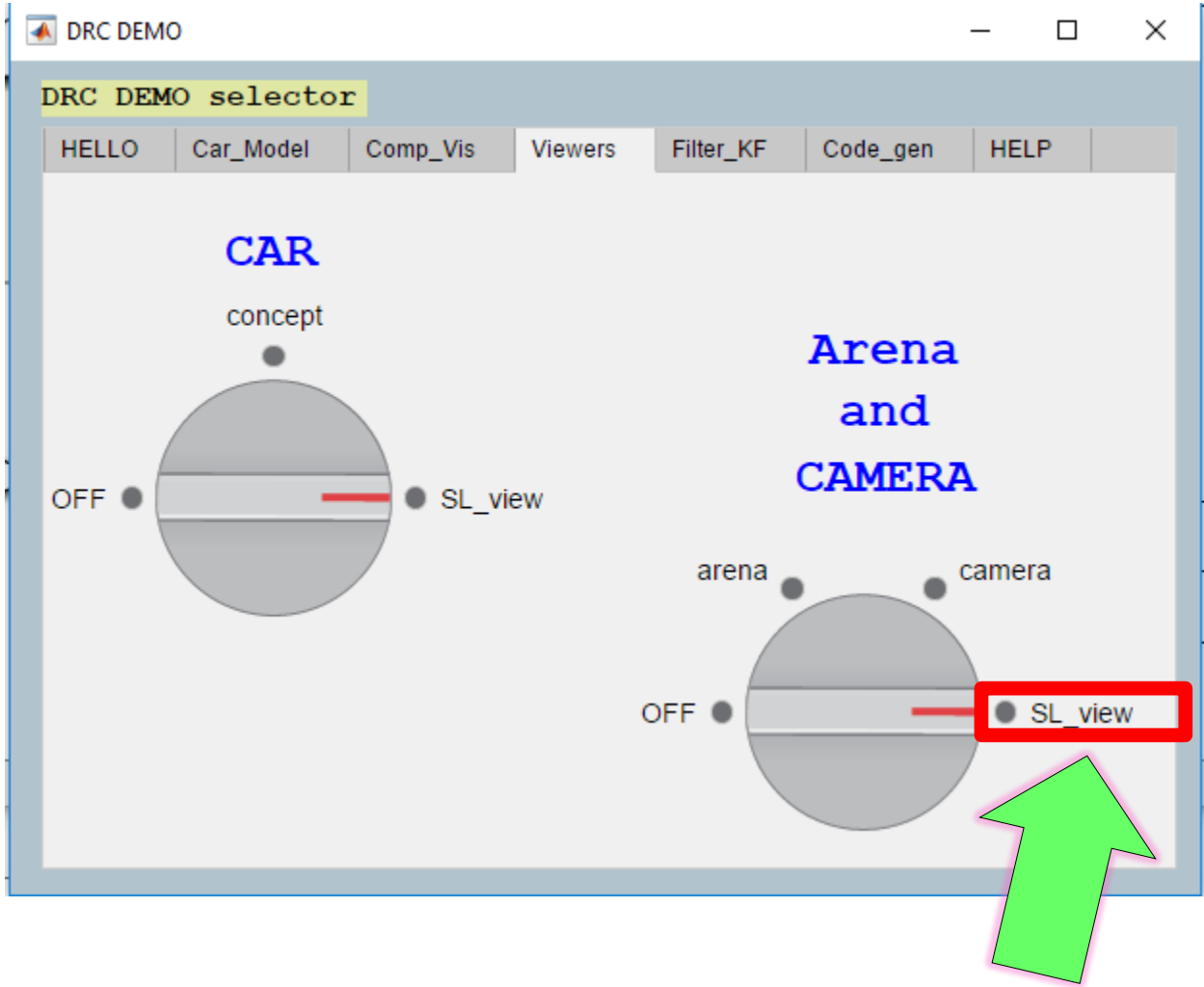
- **So what do you do with it ?**
  - **YOU** manually run the TEST script.

  - A very simple viewer

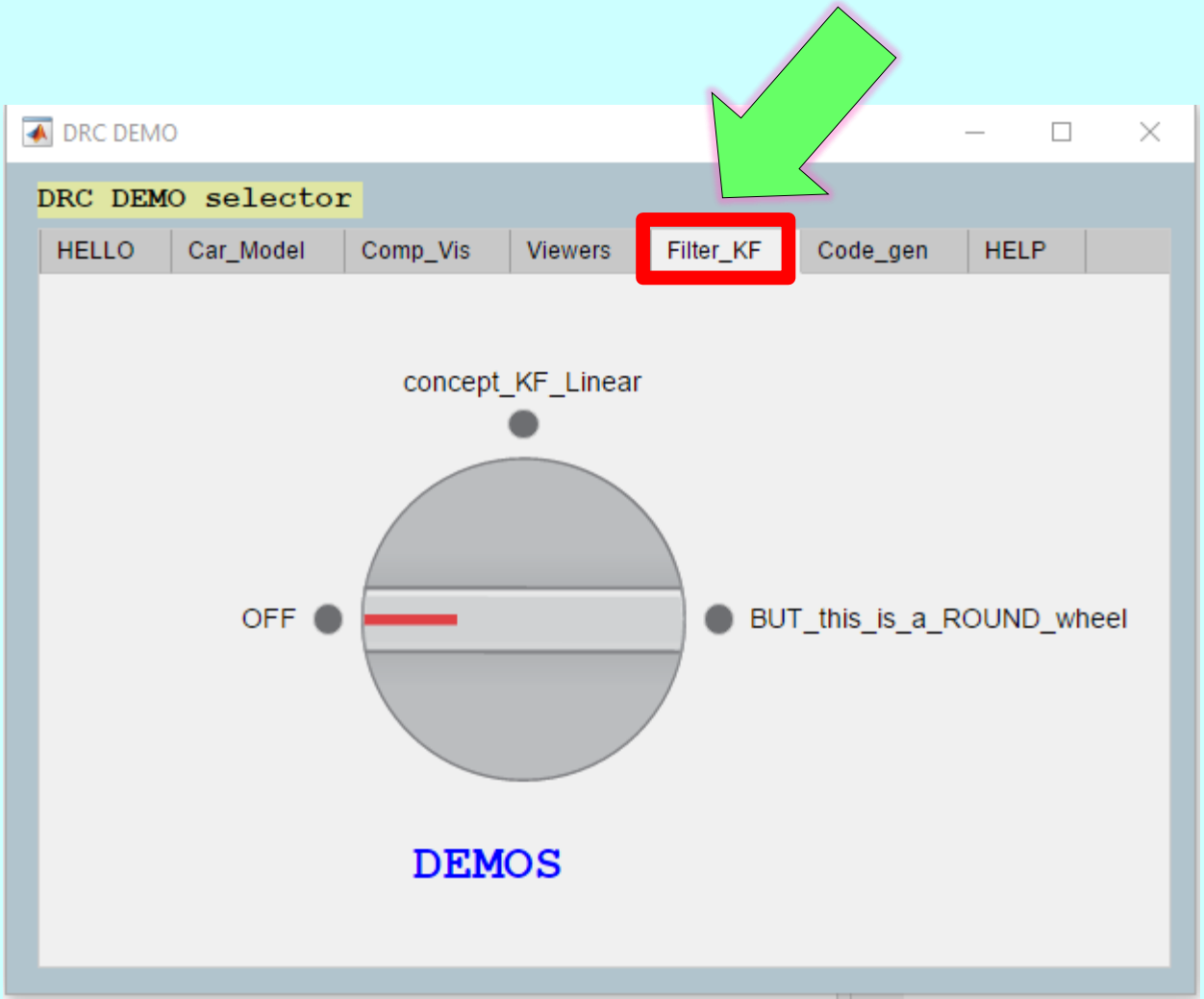


Comments on the **VIEWERS** demo: Part 5 of 5

- Opens a Simulink model
- The Simulink model contains a block that calls our
  - race track arena
  - virtual camera
- **So what do you do with it ?**
  - **YOU** manually run the model.
  - A very simple viewer



The Filter_KF



The Filter_KF: overview

- Develop a Linear Discrete time Kalman filter
  - Test with a familiar 2-dof spring mass damper system

- Emphasize that MATLAB has many, many Kalman filter functions
  - Linear
  - Extended
  - Unscented

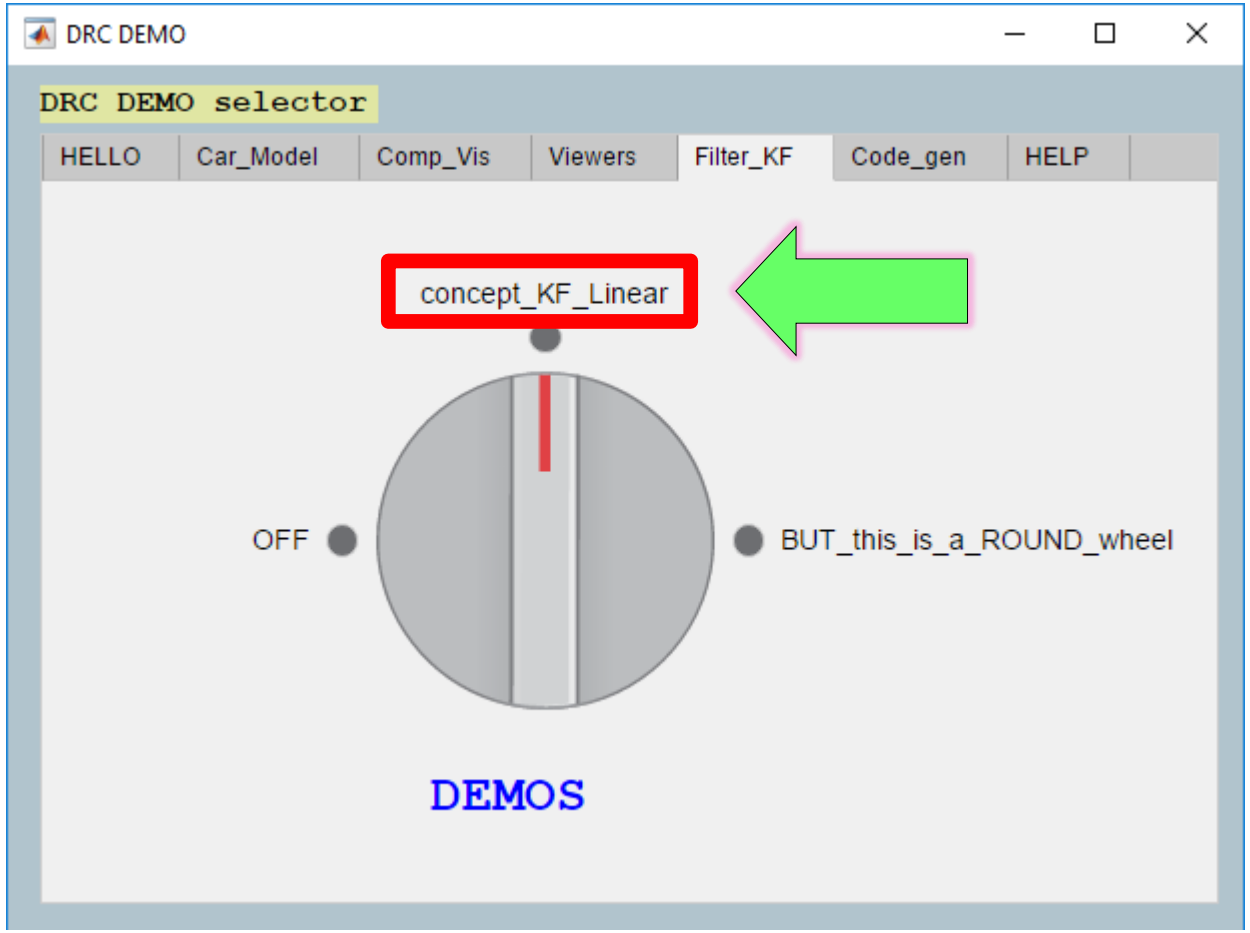
Comments on the **Filter_KF** demo: Part 1 of 2

- Opens a collection of MATLAB files
  - A main LIVE script
    - **`bh_test_vanilla_KF_2dof_mech.mlx`**
  - 1 Matlab class *.m file
    - **`my_KF_general_CLS.m`**

- The class *.m file implements a simple LINEAR discrete time Kalman filter

- The script is a TEST harness for testing the code defined by the MATLAB class.

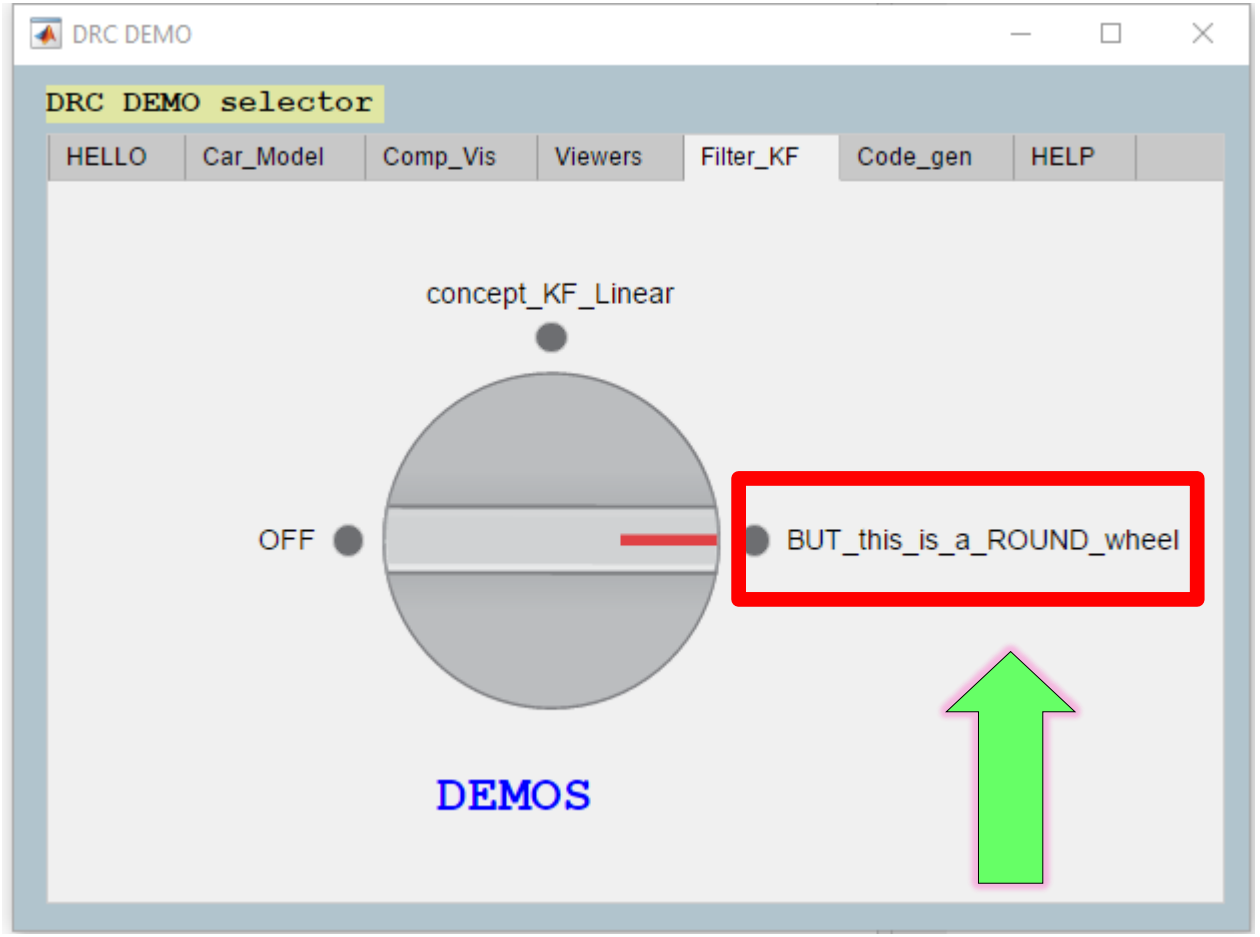
- **So what do you do with it ?**
  - **YOU** manually run the TEST script.



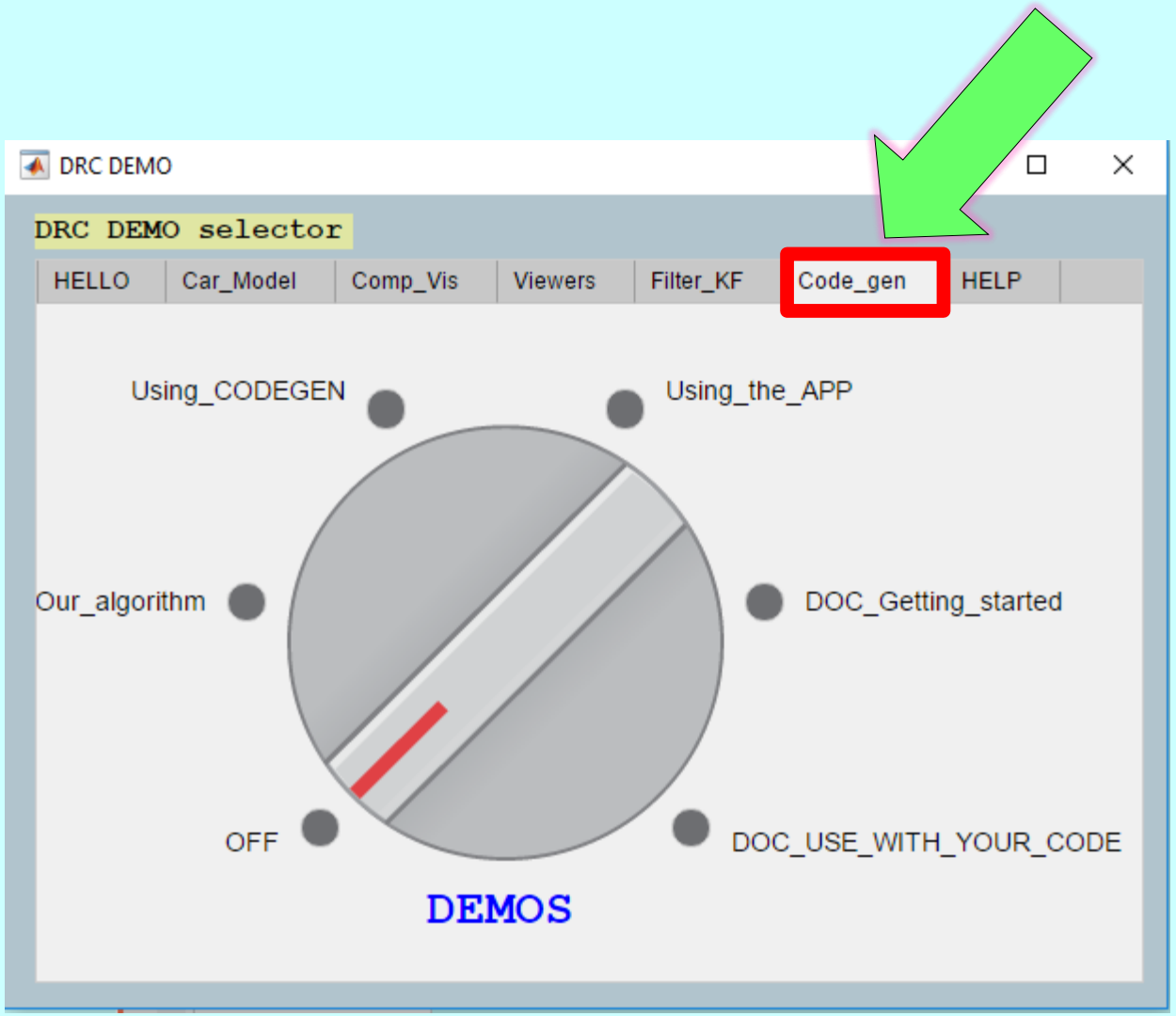
Comments on the **Filter_KF** demo: Part 2 of 2

- Opens the MATLAB Help Browser
  - Shows the many, many hits when you search for “Kalman filter”

- **So what do you do with it ?**
  - Have a look at some of the search results.



The Code_gen


DRC DEMO
DRC DEMO selector
HELLO
Car_Model
Comp_Vis
Viewers
Filter_KF
Code_gen
HELP
Using_CODEGEN
Using_the_APP
Our_algorithm
DOC_Getting_started
OFF
DOC_USE_WITH_YOUR_CODE
DEMOS

The Code_gen: overview

- Automatically convert the vision algorithm into stand alone C-code
  - Do this programmatically the `codegen()` function
  - Do this interactively using the MATLAB Coder APP

- If you've prototyped an algorithm in MATLAB …. You can automatically convert this MATLAB code into standalone C-code

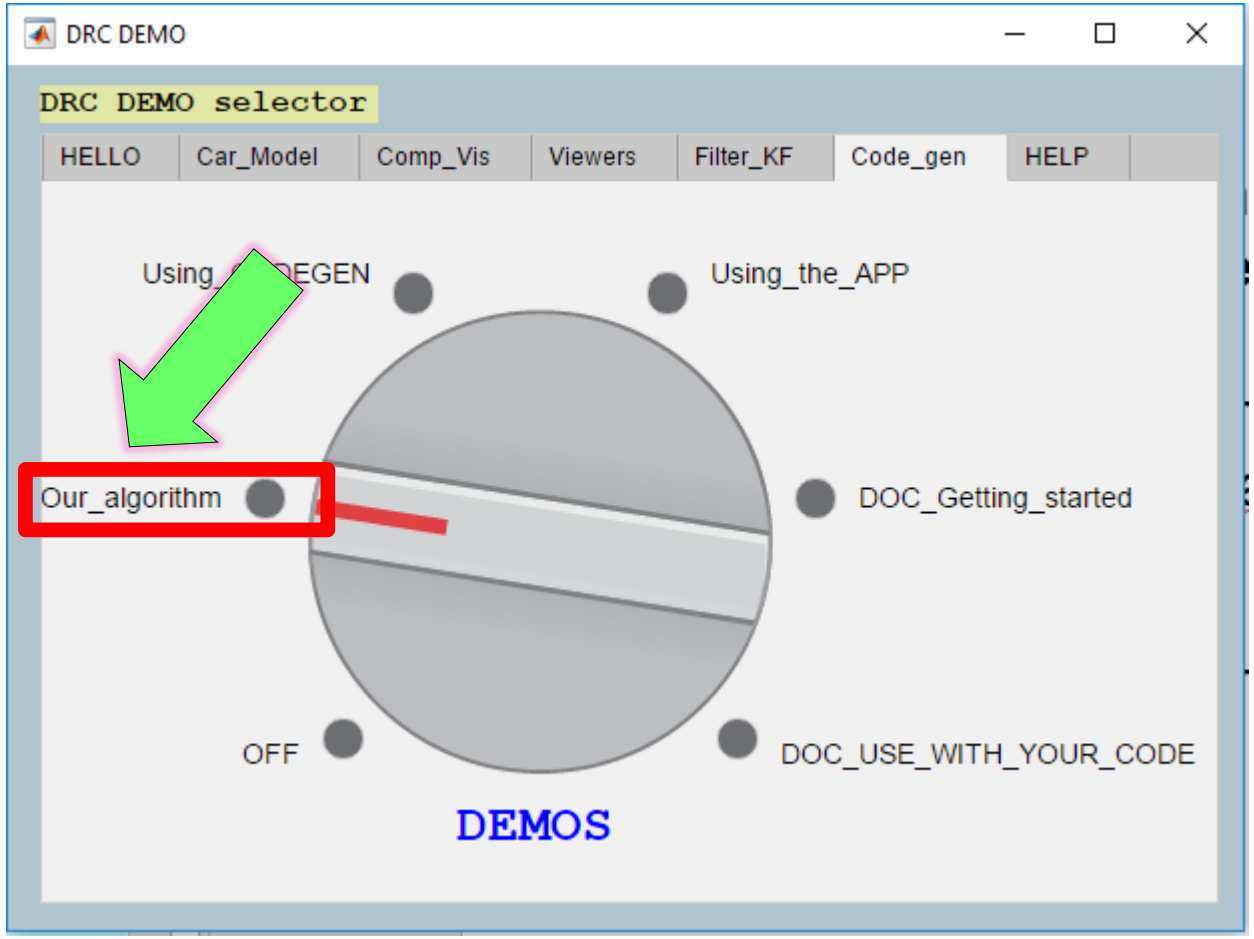
Comments on the **Code_gen** demo: Part 1 of 5

- Opens a collection 2 MATLAB files
  - A main script
    - `bh_test_harness_for_algorithm.m`
  - 1 function *.m file
    - `bh_my_detect_ALGORITHM_for_codegen.m`

- The function *.m defines an example prototype algorithm that I want to convert into stand alone C-code

- The script is a TEST harness for testing the algorithm.

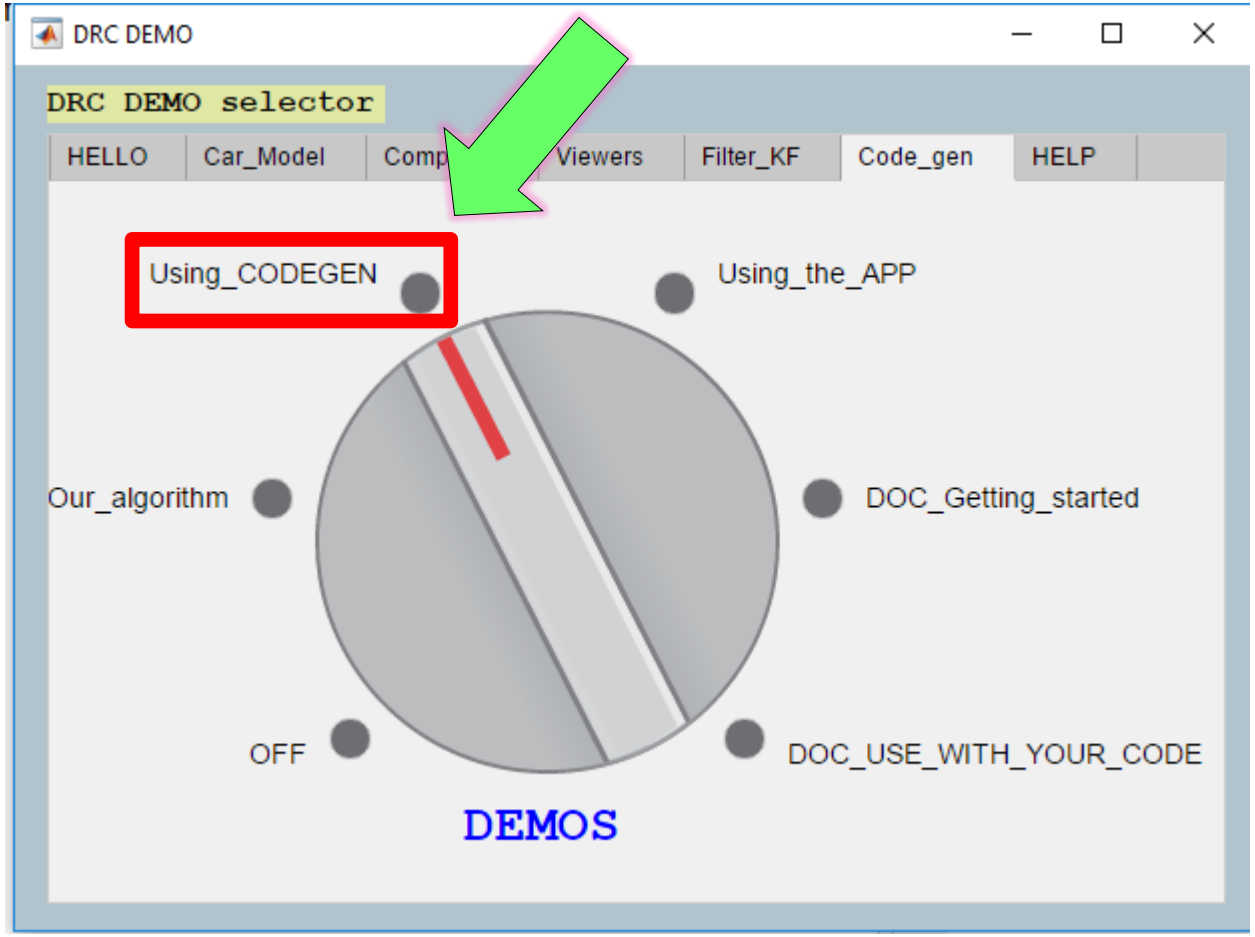
- **So what do you do with it ?**
  - **YOU** manually run the TEST script.
  - We're simply establishing that I have an algorithm AND a test harness for testing that algorithm
  - In the NEXT step we will use both of these *.m files



Comments on the **Code_gen** demo: `Part 2 of 5`

- Opens a script file
  - **`bh_do_code_generation_using_CODEGEN.m`**

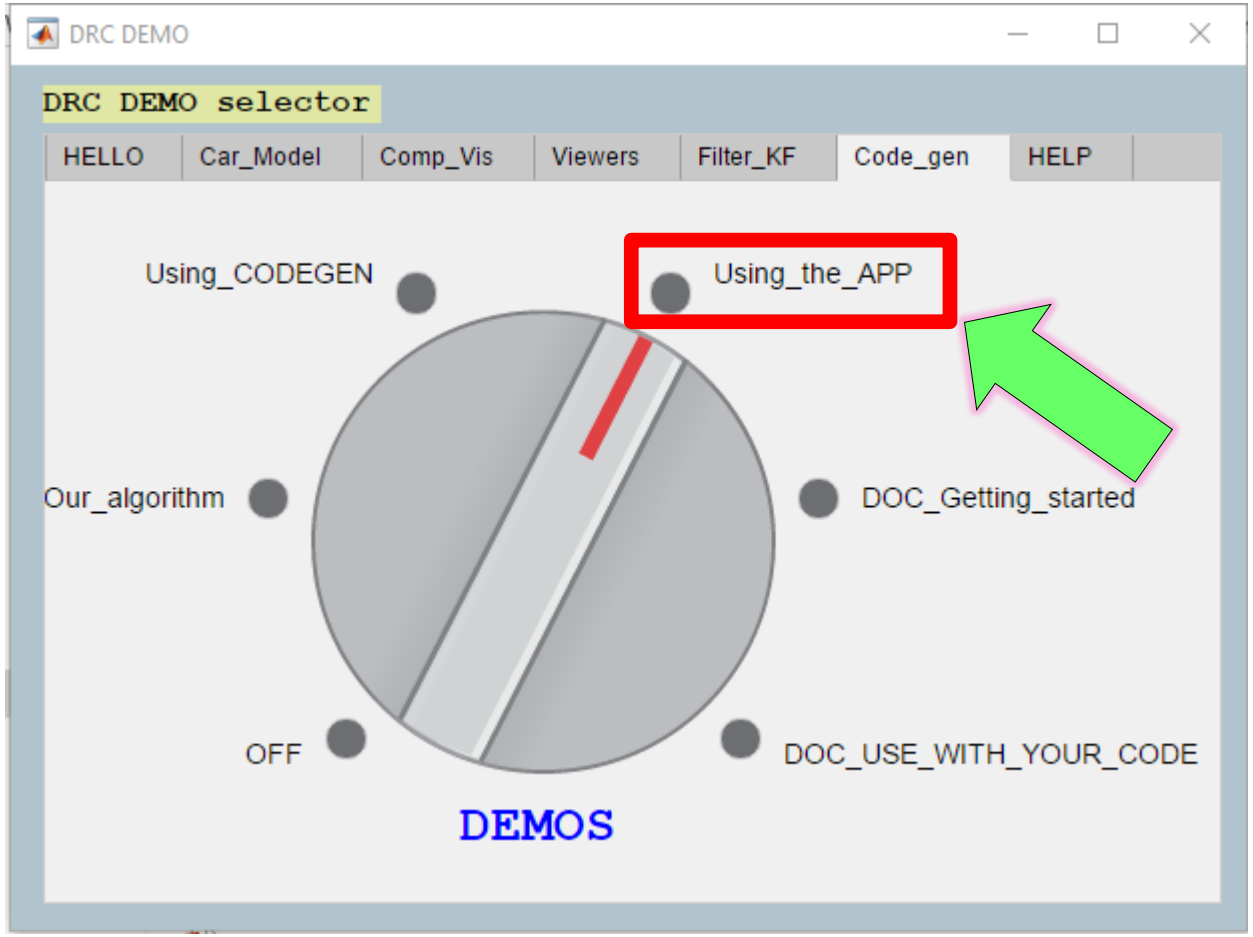
- **So what do you do with it ?**
  - **YOU** manually run the script.
  - The script shows the steps involved in converting the algorithm into stand alone C-code.
    - This is done using the `codegen()` function
  - Have a look at the code generation report showing the *.c and *.h files that have automatically been created.
  - In the next step, we'll show how this same task (of creating stand alone C-code) can also be done using an interactive APP.



Comments on the **Code_gen** demo: Part 3 of 5

- Opens a script file
  - `bh_do_code_generation_using_APP.m`

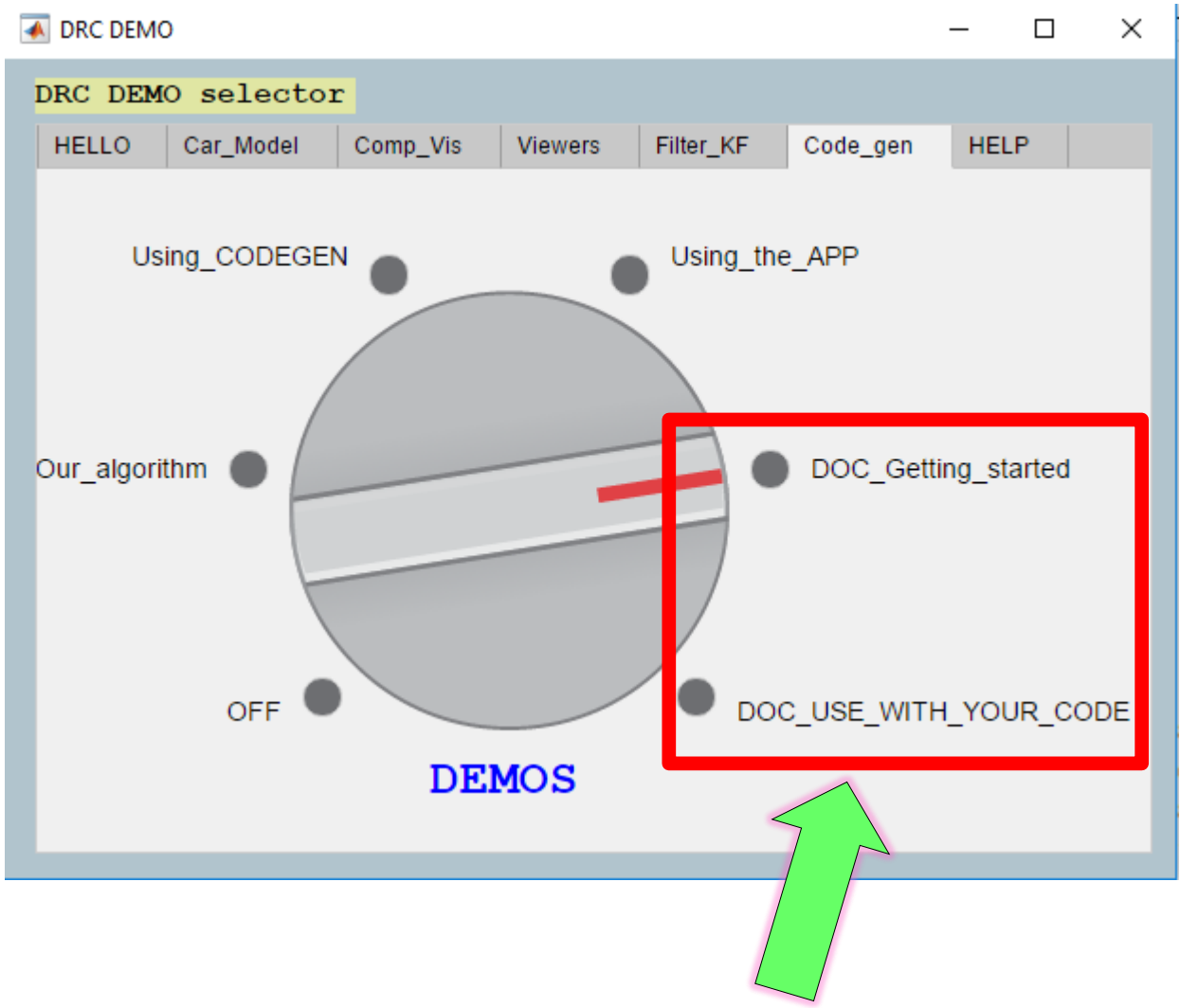
- **So what do you do with it ?**
  - **YOU** manually run the script.
  - The script launches the MATLAB Coder APP … and a premade project file.
  - Explore the app
  - In the next demo we'll show the DOC page for getting a step by step tutorial on how to drive the APP.



Comments on the **Code_gen** demo: Part 4+5 of 5

- Opens the MATLAB Help Browser
  - Shows DOC on converting MATLAB code to C-code.

- **So what do you do with it ?**
  - Have a look at the DOC and tutorials on converting MATLAB code to C-code.



END