CHAPTER 1

Matlab .m Files

1.1 minseg.m

Code Listing 1.1: [minseg.m]: Root file

```
%% [Global  ]
minseg_0p0p0p0_global

%% [Input   ]: Model
minseg_1p0p0p0_input

%% [Input   ]: Script: Commands

ui.x.build   = 0; % rebuild required for any change in [input]: model / [init]: model.
ui.x.write   = 0; % not yet implemented
ui.x.read    = 0;
ui.x.plot    = 0; % enables read/write by default.
ui.x.save    = 0; % enables read/write by default.
ui.x.cleanup = 0; % enables read/write by default.

%% [Input   ]: Script: Serial

switch 1  % serial duration
  case 0; ui.srl.n.transmits = 100; % [samples]
  case 1; ui.srl.T.transmits = 020; % [seconds]
end

%% [Input   ]: Script: Save
ui.save.label = '';

%% [Init    ]: Define parameters
minseg_2p0p0p0_init_general

minseg_2p1p0p0_init_model_general
minseg_2p1p1p0_init_model_plant
minseg_2p1p2p0_init_model_controller
minseg_2p1p3p0_init_model_io
```

```
minseg_2p1p9p0_init_model_build

minseg_2p2p0p0_init_serial_write
minseg_2p2p1p0_init_serial_read
minseg_2p2p2p0_init_serial_general
minseg_2p2p3p0_init_serial_reads
minseg_2p2p9p0_init_model_build

%minseg_2p3p0p0_init_build

%% [Process]:

% build (normal mode) / run (external mode)
if ui.x.build
minseg_3p0p0p0_process_build
end

% serial transmit (normal mode only)
if (ui.x.write || ui.x.read || ui.x.plot || ui.x.save )
minseg_3p1p0p0_process_serial_transmit
end

% serial post-processing
if (             ui.x.read || ui.x.plot || ui.x.save )
minseg_3p2p0p0_process_serial_reads

% if ui.mdl.case == 2
% minseg_3p3p1p0_process_motorTF
% end
%
% if ui.mdl.case == 3
% minseg_3p3p1p0_process_gyroBias
% end

end

%% [Output]:

% save
if ui.x.save
minseg_4p0p0p0_output_save
```

```
end

% plot
if ui.x.plot
minseg_4p1p0p0_output_serial_plot
% minseg_4p1p1p0_output_serial_plot
end

%% [Cleanup]:

if ui.x.cleanup
minseg_5p0p0p0_cleanup
end

%% End
```

Code Listing 1.1: [minseg.m]: Root file

1.1.1 Global Setup

Code Listing 1.2: [minseg.m]: Global Setup

```
%% [Global ]:
clc
clearvars
close all

% close all loaded simulink models and libraries.
% close_system( find_system('SearchDepth', 0) )

% close and delete all serial connections
if ~isempty(instrfindall)
  fclose   (instrfindall);
  delete   (instrfindall);
end

%% [Global ]: Add subdirectories to Matlab path
root.dir     = cd;
root.sub.dir = { [root.dir '/1. General Tools/'              ]
                 [root.dir '/1. General Tools/0. Bessel Poles' ]
                 [root.dir '/1. General Tools/1. fftPlus'     ]
                 [root.dir '/1. Subscripts'                   ]
                 [root.dir '/2. Model metadata'               ]
                 [root.dir '/3. Data'                         ]
               };

root.n.sub.dir = size( root.sub.dir, 1 );
for i0 = 1 : root.n.sub.dir
  addpath(   root.  sub.dir{ root.n.sub.dir - (i0 - 1), 1 } )
end

%% [Global ]: Add subdirectories to Simulink path
Simulink.fileGenControl( 'set'                                                 ...
                       , 'CacheFolder',   [ root.dir '/2. Model metadata/Work' ] ...
                       , 'CodeGenFolder', [ root.dir '/2. Model metadata/Code' ] ...
                       , 'createDir',       true                                ...
                       )

%% End
```

Code Listing 1.2: [minseg.m]: Global Setup

1.1.2 User Inputs

Code Listing 1.3: [minseg.m]: User Inputs

```
%% [Input   ]: Model: General
ui.mdl.label = 'minseg_M2V3_2017a';

ui.mdl.mode  = 0;
% 0: normal
% 1: external

ui.mdl.case  = 0;
%  ##: Case Description:      Plant:    Controller:    Command:
% -01: Clear board            Empty     Empty          Empty    % not yet implemented
% +00: Custom                 Custom    Custom         Custom
% +01: Motor characterization Hardware  FF  - v.motor  0 --> 10 [V]
% +02: Gyro bias calibration  Hardware  PID - w.motor  0 --> 00 [rad/s]

%% [Input   ]: Model: Plant

ui.plant.dynamics.mode        = 0;
% 0: actual hardware
% 1: simulated dynamics (non-linear)
% 2: simulated dynamics (    linear)

ui.plant.n.batteries          = 6; % [range: 0 - 6]

ui.plant.x.bluetoothModule    = 1;
% 0: bluetooth module not inserted into board.
% 1: bluetooth module     inserted into board.

ui.plant.supply.mode          = 1;
% 0: 9.00 [V] (battery pack)
% 1: 4.50 [V] (usb cable)
% Important: Do NOT set to usb power if actually using battery power.

%% [Input   ]: Model: Controller: body.pitch.theta
% not yet implemented.

ui.ctrl.body.pitch.theta.mode = 0;
% #: mode:          input command:    [input command unit]:

```

```
%% [Input  ]: Model: Controller: motor.v

ui.ctrl.motor_v.mode          = 1;
% #: mode:          input command:    [input command unit]:
% 0: feedForward    v.motor           [V]
% 1: PID            w.motor           [rad/s]

switch ui.ctrl.motor_v.mode

case 0 % feed forward (input: v.motor)
  ui.io.write.ctrl.motor_v.cmd.tStart          (1,1) =  0;
  ui.io.write.ctrl.motor_v.cmd.val.x           (1,1) = +10;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.01;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -0.01;

case 1 % PID          (input: w.motor)
  ui.io.write.ctrl.motor_v.cmd.tStart          (1,1) =  0;
  ui.io.write.ctrl.motor_v.cmd.val.x           (1,1) =  0.50 * 2*pi;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max(1,1) = +0.10;
  ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min(1,1) = -inf;

end

%% [Input  ]: Model: Serial
ui.srl.mode.address = 0;
% 0: left      usb port (2015 Macbook Pro)
% 1: left-rear usb port (2008 Macbook Pro)

% Note: Needs to be changed manually for external mode:
% Simulink: Configuration parameters: Hardware implementation: Host-board connection

ui.srl.T.decimation = 0; % [integer] [default: 0]
% Integer factor of board sample time (mdl.T.sample)
%  in which to iterate serial processes.
% If 0, minimum possible value will be used.
% (Could be greater than 1 if combined size of reads/writes is sufficiently large.)

%% End
```

Code Listing 1.3: [minseg.m]: User Inputs

1.1.3 Initialization

1.1.3.1 General

Code Listing 1.4: [minseg.m]: Initialization - General

```
%% [Init    ]: Conversions
k.intmax.uint8 = double( intmax('uint8') );
k.intmax.int16 = double( intmax('int16') );

k_deg2rad      = 2*pi / 360;
k_rad2deg      = 1 / k_deg2rad;

k.byte2bit     = 8;
k.bit2byte     = 1 / k.byte2bit;

k.lb2kg        = 0.45359233;
k.kg2lb        = 1 / k.lb2kg;

k.in2m         = 0.0000254;
k.m2in         = 1 / k.in2m;

%% End
```

Code Listing 1.4: [minseg.m]: Initialization - General

1.1.3.2 Model

1.1.3.2.1 General

Code Listing 1.5: [minseg.m]: Initialization - Model - General

```
%% [Init    ]: Initialize user-defined parameters
mdl.label= ui.mdl.label;
mdl.mode = ui.mdl.mode;
mdl.case = ui.mdl.case;

%% [Init    ]: Load model, if not already loaded
if ~bdIsLoaded( mdl.label )
load_system(    mdl.label );
end

%% [Init    ]: Define general model parameters

mdl.object = get_param(mdl.label, 'Object');

switch mdl.mode
  case 0; mdl.T.sample = 0.005; % 0: normal
  case 1; mdl.T.sample = 0.030; % 1: external
end

%% End
```

Code Listing 1.5: [minseg.m]: Initialization - Model - General

1.1.3.2.2 Plant

Code Listing 1.6: [minseg.m]: Initialization - Model - Plant

```
%% [Init   ]: Initialize user-defined parameters
plant.supply.  mode     = ui.plant.supply.  mode;
plant.dynamics.mode     = ui.plant.dynamics.mode;

plant.n.batteries       = ui.plant.n.batteries;
plant.x.bluetoothModule = ui.plant.x.bluetoothModule;

%% [Init   ]: Define general plant parameters

switch plant.supply.mode
  case 0; plant.supply.v = 9.00; % [V]
  case 1; plant.supply.v = 4.50; % [V]
end

a.gravity                = 9.81; % acceleration [m / s^2]

load( 'bessel poles.mat' )

%% [Init   ]: Verify legitimate operating modes

if plant.supply.mode == 0
assert( plant.n.batteries == 6,                                        ...
      [                                                                ...
       'Battery power is enabled (plant.supply.mode == 0);\n'          ...
       'however, the number of batteries in use is not equal to\n'     ...
       'the number of batteries needed to operate in '                 ...
       'battery power mode (plant.n.batteries ~= 6)'                   ...
      ]                                                                ...
      );
end

%% [Init   ]: Define parameters based on user-specified plant dynamics

switch plant.dynamics.mode
  case 0; minseg_2p1p1p1_init_model_plant_hardware
  case 1; minseg_2p1p1p2_init_model_plant_nonlinearDynamics
  case 2; minseg_2p1p1p3_init_model_plant_linearDynamics
end
```

```

%% End
```

Code Listing 1.6: [minseg.m]: Initialization - Model - Plant

1.1.3.2.2.1 Hardware

Code Listing 1.7: [minseg.m]: Initialization - Model - Plant - Hardware

```
%% [Init   ]: Motor: Driver
mtr.driver. left.  pin.pos = 6;
mtr.driver. left.  pin.neg = 8;
mtr.driver. middle.pin.pos = 2;
mtr.driver. middle.pin.neg = 5;

%% [Init   ]: Motor: Encoder
% not yet implemented
% mask encoder model, then use pins as mask parameters
mtr.encoder.left.  pin.A   = 19;
mtr.encoder.left.  pin.B   = 18;
mtr.encoder.middle.pin.A   = 15;
mtr.encoder.middle.pin.B   = 62;

mtr.encoder.countPerRev    = 720;
mtr.encoder.radPerRev      = 2 * pi;

%% [Init   ]: Motor: Encoder: angVel bessel filter: design parameters
mtr.encoder.filter.T.settle = mdl.T.sample * 25; % [s]
mtr.encoder.filter.order    = 4;     % [-] [integer] [ range: 02 : 10 ]

%% [Init   ]: Motor: Encoder: angVel bessel filter: transfer function
% divide normalize poles by settling time
mtr.encoder.filter.s.poles = poly( s.pole.bessel{mtr.encoder.filter.order} ...
                                 / mtr.encoder.filter.T.settle             ...
                                 );



% create transfer function
mtr.encoder.filter.s.tf    =   tf( mtr.encoder.filter.s.poles(end)         ...
                                 , mtr.encoder.filter.s.poles              ...
                                 );



% discretize transfer function
mtr.encoder.filter.z.tf    =  c2d( mtr.encoder.filter.s.tf                 ...
                                 , mdl.T.sample                            ...
                                 );
```

```

% break transfer function into numerator and demonintor polynomials
[ mtr.encoder.filter.s.num ...
, mtr.encoder.filter.s.den ...
] = tfdata                 ...
( mtr.encoder.filter.s.tf  ...
);

[ mtr.encoder.filter.z.num ...
, mtr.encoder.filter.z.den ...
] = tfdata                 ...
( mtr.encoder.filter.z.tf  ...
);

% convert cells to matrices
mtr.encoder.filter.s.num = mtr.encoder.filter.s.num{:};
mtr.encoder.filter.s.den = mtr.encoder.filter.s.den{:};
mtr.encoder.filter.z.num = mtr.encoder.filter.z.num{:};
mtr.encoder.filter.z.den = mtr.encoder.filter.z.den{:};

%% [Init   ]: Motor: Encoder: angVel bessel filter: state-space

% create s-plane state space equations (canonical representation)
mtr.encoder.filter.s.ss.A        = diag( ones( mtr.encoder.filter.order - 1, 1 ), 1);
mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.poles( end : -1 : 2 );
mtr.encoder.filter.s.ss.A(end,:) = mtr.encoder.filter.s.ss.A(end,:) ...
                                 / mtr.encoder.filter.s.poles( 1 ) * -1;

mtr.encoder.filter.s.ss.B        = [ zeros(    mtr.encoder.filter.order - 1, 1 ); 1 ];
mtr.encoder.filter.s.ss.C        = [ zeros( 1, mtr.encoder.filter.order - 1    )  1 ];
mtr.encoder.filter.s.ss.D        = 0;


% discretize s-plane state space equations (canonical representation)
[ mtr.encoder.filter.z.ss.A ... phi
, mtr.encoder.filter.z.ss.B ... gamma
] = zohe                    ...
( mtr.encoder.filter.s.ss.A ... A
, mtr.encoder.filter.s.ss.B ... B
, mdl.T.sample                 ... T
);

mtr.encoder.filter.z.ss.C         = mtr.encoder.filter.s.ss.C;
mtr.encoder.filter.z.ss.D         = mtr.encoder.filter.s.ss.D;

%% [Init    ]: Gyroscope
gyro.dlpf.mode   = 0;  % [ default: 0 ]
% | # | maxValue [deg/s] | bandwidth [Hz] | delay [s] |
% | 0 | +/- 0250         | 256            | 00.98     |
% | 1 | +/- 0500         | 188            | 01.90     |
% | 2 | +/- 1000         | 098            | 02.80     |
% | 3 | +/- 2000         | 042            | 04.80     |
% | 4 | +/- ????         | 020            | 08.30     |
% | 5 | +/- ????         | 010            | 13.40     |
% | 6 | +/- ????         | 005            | 18.60     |

switch gyro.dlpf.mode
case 0; gyro.maxVal = 0250 * k_deg2rad;
case 1; gyro.maxVal = 0500 * k_deg2rad;
case 2; gyro.maxVal = 1000 * k_deg2rad;
case 3; gyro.maxVal = 2000 * k_deg2rad;
end

gyro.k_raw2actual = gyro.maxVal / k.intmax.int16;

% [source: 1. Test Cases/1. Gyro Bias Calibration]
gyro.x.bias      = -266.0779700;
gyro.y.bias      = -135.5037500;
gyro.z.bias      = -034.3493271;

gyro.x.reset     =  0;
gyro.y.reset     =  0;
gyro.z.reset     =  0;

%% [Init    ]: Gyroscope      : angVel bessel filter: design parameters
gyro.filter.T.settle = mdl.T.sample * 25; % [s]
gyro.filter.order    = 4;      % [-] [integer] [ range: 02 : 10 ]

%% [Init    ]: Gyroscope      : angVel bessel filter: transfer function
% divide normalize poles by settling time
```

```
gyro.filter.s.poles = poly( s.pole.bessel{gyro.filter.order} ...
                          / gyro.filter.T.settle             ...
                          );


% create transfer function
gyro.filter.s.tf    =   tf( gyro.filter.s.poles(end)         ...
                          , gyro.filter.s.poles              ...
                          );


% discretize transfer function
gyro.filter.z.tf    =  c2d( gyro.filter.s.tf                 ...
                          , mdl.T.sample                     ...
                          );

% break transfer function into numerator and demonintor polynomials
[ gyro.filter.s.num ...
, gyro.filter.s.den ...
] = tfdata          ...
( gyro.filter.s.tf  ...
);

[ gyro.filter.z.num ...
, gyro.filter.z.den ...
] = tfdata          ...
( gyro.filter.z.tf  ...
);

% convert cells to matrices
gyro.filter.s.num = gyro.filter.s.num{:};
gyro.filter.s.den = gyro.filter.s.den{:};
gyro.filter.z.num = gyro.filter.z.num{:};
gyro.filter.z.den = gyro.filter.z.den{:};

%% [Init    ]: Gyroscope     : angVel bessel filter: state-space

% create s-plane state space equations (canonical representation)
gyro.filter.s.ss.A        = diag( ones( gyro.filter.order - 1, 1 ), 1);
gyro.filter.s.ss.A(end,:) = gyro.filter.s.poles( end : -1 : 2 );
gyro.filter.s.ss.A(end,:) = gyro.filter.s.ss.A(end,:) ...
```

```
                            / gyro.filter.s.poles( 1 ) * -1;

gyro.filter.s.ss.B          = [ zeros(     gyro.filter.order - 1, 1 ); 1 ];
gyro.filter.s.ss.C          = [ zeros( 1, gyro.filter.order - 1     )  1 ];
gyro.filter.s.ss.D          = 0;


% discretize s-plane state space equations (canonical representation)
[ gyro.filter.z.ss.A ... phi
, gyro.filter.z.ss.B ... gamma
] = zohe                ...
( gyro.filter.s.ss.A ... A
, gyro.filter.s.ss.B ... B
, mdl.T.sample       ... T
);

gyro.filter.z.ss.C          = gyro.filter.s.ss.C;
gyro.filter.z.ss.D          = gyro.filter.s.ss.D;

%% [Init   ]: Accelerometer
accel.afs_sel.mode   = 0;  % [ Required: 0 ]
% | # | maxValue [g] | Sensitivity [LSB/mg] |
% | 0 | +/- 02       | 8192
% | 1 | +/- 04       | 4096
% | 2 | +/- 08       | 2048
% | 3 | +/- 16       | 1024

assert( accel.afs_sel.mode == 0 );

switch accel.afs_sel.mode
case 0; accel.maxVal = 02 * a.gravity;
case 1; accel.maxVal = 04 * a.gravity;
case 2; accel.maxVal = 08 * a.gravity;
case 3; accel.maxVal = 16 * a.gravity;
end

accel.k_raw2actual = accel.maxVal / k.intmax.int16;

%% End
```

Code Listing 1.7: [minseg.m]: Initialization - Model - Plant - Hardware

1.1.3.2.2.2 Nonlinear Dynamics Model

Code Listing 1.8: [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model

```
%% End
```

Code Listing 1.8: [minseg.m]: Initialization - Model - Plant - Nonlinear Dynamics Model

1.1.3.2.2.3 Linear Dynamics Model

Code Listing 1.9: [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model

```
%% [Init   ]: Plant: Wheel   (single)

% mass measurement precision: 0.01 lb
% note: this could be improved with a better scale.

plant.axel.m         = 0.000;     %                          [kg]       [note low precision.]

plant.wheel.r        = 0.021;     %  radius                  [m]        [source: howard]
plant.wheel.m        = 0.036 / 2; % (includes axel)          [kg]       [source: howard]
plant.wheel.J        = 7.460e-6;  %  moment of inertia       [kg / m^2] [source: howard]
                                  %  measured from center of mass of wheel

%% [Init   ]: Plant: Body: Masses
% note: body does not include wheels.

% mass measurement precision: 0.01 lb
% note: this could be improved with a better scale.

% plant.board.     m   = 1.000 * k.lb2kg; %                   [kg]
% plant.motorCable.m   = 0.010 * k.lb2kg; % (quantity: 1)     [kg] [note low precision.]
% plant.motor.     m   = 0.220 * k.lb2kg; % (quantity: 1)     [kg]
% plant.battery.   m   = 1.000 * k.lb2kg; % (quantity: 1)     [kg]

% plant.bluetooth. m   = 0.000 * k.lb2kg; % bluetooth module  [kg] [note low precision.]
% plant.usbCable.  m   = 0.040 * k.lb2kg; % (not included)    [kg]

% plant.body.m         =  plant.board.m                                 ...
%                      +  plant.motor.m      * 2                        ...
%                      +  plant.motorCable.m * 2                        ...
%                      +  plant.battery.m    * plant.n.battery          ...
%                      +  plant.bluetooth.m  * plant.x.bluetoothModule;
%                              %  mass   [kg]
```

```

plant.body.m         =  1.030; % (not included)    [kg]
                               % net measurement taken to reduce rounding errors.
                               % [taken with 6 batteries].

%% [Init    ]: Plant: Body
% note: does not include wheels.

plant.body.l.h       =  8.00 * k.in2m; %  height  [m]
plant.body.l.w       =  3.25 * k.in2m; %  width   [m]
plant.body.l.d       =  2.50 * k.in2m; %  depth   [m]

switch plant.x.bluetoothModule

case 0 % Not inserted
  switch plant.n.batteries
  case 0; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 5; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 6; plant.body.f.natural =  1; %  natural frequency [rad/s]
  end

case 1 % Inserted
  switch plant.n.batteries
  case 0; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 5; plant.body.f.natural =  1; %  natural frequency [rad/s]
  case 6; plant.body.f.natural =  3.5087719; %  natural frequency [rad/s]
  end

end

plant.body.w.natural = 2 * pi * plant.body.f.natural;
                           %  natural angular frequency [rad/s]

plant.body.l.c       =  3 * (a.gravity - plant.body.w.natural^2 * plant.wheel.r) ...
                     / (4 *              plant.body.w.natural^2                );
                           %  wheel axel to center of mass of robot [m]

plant.body.J.x       =  plant.body.m *  plant.body.l.c^2                   ...
                     /  3;
                           %  moment of inertia (pitch) [kg / m^2]
                           % (measured from center of mass of robot)
```

```

plant.body.J.y      = plant.body.m                                  ...
                    * (plant.body.l.w^2 + plant.body.l.d^2)         ...
                    / 12;
                         % moment of inertia (yaw)   [kg / m^2]
                         % (measured from center of mass of robot)

%% [Init    ]: Plant: Net (body + 2 * wheel)
plant.net.m         = plant.body.m + 2 * plant.wheel.m; % [kg]

%% [Init    ]: Plant: Motor
mtr.R               = 4.400; % resistance          [ohm     ] [source: howard]
mtr.k.dlambda       = 0.495; % back EMF constant   [V*s / rad ] [source: howard]
mtr.k.torque        = 0.470; % torque  constant    [N*m / A   ] [source: howard]

switch plant.x.bluetoothModule

case 0 % Not inserted
  switch plant.n.batteries
  case 0
  mtr.k.v2w         = 1.000; % transfer function (y/u) [rad / (s*V)]
                             % (measured when body is upright AND
                             % both wheels are at equivalent speed
                             % in a common direction.)

  case 5
  mtr.k.v2w         = 1.000; % transfer function (y/u) [rad / (s*V)]
                             % (measured when body is upright AND
                             % both wheels are at equivalent speed
                             % in a common direction.)

  case 6
  mtr.k.v2w         = 1.000; % transfer function (y/u) [rad / (s*V)]
                             % (measured when body is upright AND
                             % both wheels are at equivalent speed
                             % in a common direction.)

  end
```

```

case 1 % Inserted
  switch plant.n.batteries
  case 0
  mtr.k.v2w          =  1.000; %  transfer function (y/u) [rad / (s*V)]
                               % (measured when body is upright AND
                               %  both wheels are at equivalent speed
                               %  in a common direction.)

  case 5
  mtr.k.v2w          =  1.000; %  transfer function (y/u) [rad / (s*V)]
                               % (measured when body is upright AND
                               %  both wheels are at equivalent speed
                               %  in a common direction.)

  case 6
  mtr.k.v2w          =  3/3.35; %  transfer function (y/u) [rad / (s*V)]
                               % (measured when body is upright AND
                               %  both wheels are at equivalent speed
                               %  in a common direction.)

  end

end

mtr.k.friction       =  mtr.k.torque * (1 - mtr.k.dlambda * mtr.k.v2w)     ...
                     / (mtr.R * mtr.k.v2w);
                           %  coefficient of friction [-          ]

%% [Init   ]: Plant: State space model term abbreviations

% wheel.theta and body.theta.x (pitch) (psi)
plant.q(1,1) = plant.net. m * plant.wheel.r^2                 + plant.wheel.J;
plant.q(2,1) = plant.body.m * plant.wheel.r^2 * plant.body. l.c                ;
plant.q(3,1) = plant.body.m *                   plant.body. l.c^2 + plant.wheel.J;
plant.q(4,1) = mtr.k.torque * mtr.k.dlambda / mtr.R + mtr.k.friction           ;
plant.q(5,1) = plant.body.m * a.gravity      * plant.body. l.c                ;
plant.q(6,1) = mtr.k.torque                 / mtr.R                           ;

plant.Q{1,1} =      [ +plant.q(1) +plant.q(2)
                      +plant.q(2) +plant.q(3) ];
```

```
plant.Q{2,1} = 2 * [ +plant.q(4) -plant.q(4)
                     -plant.q(4) +plant.q(4) ];
plant.Q{3,1} =     [ +0          +0
                     +0          -plant.q(5) ];
plant.Q{4,1} =     [ +plant.q(6) +plant.q(6)
                     -plant.q(6) -plant.q(6) ];

% body.theta.y (yaw) (phi)
plant.r(1,1) =  plant.body.l.w / plant.wheel.r;

plant.R{1,1} =  0.5 * plant.wheel.m  * plant.body.l.w^2                    ...
             +  plant.body.J.y                                             ...
             +  0.5 * plant.r(1)^2 * plant.wheel.J;
plant.R{2,1} =  0.5 * plant.r(1)^2 * plant.q(4);
plant.R{3,1} =  0.5 * plant.r(1)   * mtr.k.torque / mtr.R;

% overall
plant.a{1,1} =  - plant.Q{1} \ plant.Q{3};
plant.a{2,1} =  - plant.Q{1} \ plant.Q{2};
plant.a{3,1} =  - plant.R{1} \ plant.R{2}; % note the backslash.

plant.b{1,1} =  + plant.Q{1} \ plant.Q{4};
plant.b{2,1} =  + plant.R{1} \ plant.R{3};

%% [Init   ]: Plant State Space Model: A

plant.A(1,1) =  0;
plant.A(1,2) =  0;
plant.A(1,3) =  1;
plant.A(1,4) =  0;
plant.A(1,5) =  0;
plant.A(1,6) =  0;

plant.A(2,1) =  0;
plant.A(2,2) =  0;
plant.A(2,3) =  0;
plant.A(2,4) =  1;
plant.A(2,5) =  0;
plant.A(2,6) =  0;

plant.A(3,1) =  plant.a{1}(1,1);
plant.A(3,2) =  plant.a{1}(1,2);
plant.A(3,3) =  plant.a{2}(1,1);
plant.A(3,4) =  plant.a{2}(1,2);
plant.A(3,5) =  0;
plant.A(3,6) =  0;

plant.A(4,1) =  plant.a{1}(2,1);
plant.A(4,2) =  plant.a{1}(2,2);
plant.A(4,3) =  plant.a{2}(2,1);
plant.A(4,4) =  plant.a{2}(2,2);
plant.A(4,5) =  0;
plant.A(4,6) =  0;

plant.A(5,1) =  0;
plant.A(5,2) =  0;
plant.A(5,3) =  0;
plant.A(5,4) =  0;
plant.A(5,5) =  0;
plant.A(5,6) =  1;

plant.A(6,1) =  0;
plant.A(6,2) =  0;
plant.A(6,3) =  0;
plant.A(6,4) =  0;
plant.A(6,5) =  0;
plant.A(6,6) =  plant.a{3};

%% [Init    ]: Plant State Space Model: B

plant.B(1,1) =  0;
plant.B(2,1) =  0;
plant.B(3,1) =  plant.b{1}(1,1);
plant.B(4,1) =  plant.b{1}(2,1);
plant.B(5,1) =  0;
plant.B(6,1) = -plant.b{2};

plant.B(1,2) =  0;
plant.B(2,2) =  0;
plant.B(3,2) =  plant.b{1}(1,2);
plant.B(4,2) =  plant.b{1}(2,2);
plant.B(5,2) =  0;
plant.B(6,2) = +plant.b{2};

%% [Init    ]: Plant State Space Model: C, D

plant.C      = eye  ( size( plant.A    )                      );
plant.D      = zeros( size( plant.A, 1 ), size( plant.C, 2 ) );

%% End
```

Code Listing 1.9: [minseg.m]: Initialization - Model - Plant - Linear Dynamics Model

1.1.3.2.3 Controller

Code Listing 1.10: [minseg.m]: Initialization - Model - Controller

```
%% [Init    ]: Initialize user-defined parameters
ctrl.motor_v.mode = ui.ctrl.motor_v.mode;

%% [Init    ]: Setup controller variant subsystems
ctrl.motor_v.ff. motor_v.var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
ctrl.motor_v.pid.motor_w.var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );

%% [Init    ]: Define controller model parameters

switch ctrl.motor_v.mode

case 0 %

case 1
  ctrl.motor_v.pid.motor_w.k.p = 0.500;
  ctrl.motor_v.pid.motor_w.k.i = 1.000;
  ctrl.motor_v.pid.motor_w.k.d = 0.000;

  ctrl.motor_v.pid.motor_w.int.maxVal = +plant.supply.v;
  ctrl.motor_v.pid.motor_w.int.minVal = -plant.supply.v;

end

%% End
```

Code Listing 1.10: [minseg.m]: Initialization - Model - Controller

1.1.3.2.4 Board Inputs and Outputs

Code Listing 1.11: [minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs

```
%% [Init    ]: Setup board i/o variant subsystems

% general

io.write.serial.                                  var = Simulink.Variant( 'mdl_mode == 0' );
io.write.scopes.                                  var = Simulink.Variant( 'mdl_mode == 1' );

% plant: hardware

io.write.serial.hardware.                         var = Simulink.Variant( 'plant_dynamics_mode == 0' );

io.write.serial.hardware.ff.                      var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
io.write.serial.hardware.pid.                     var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );

io.write.serial.hardware.ff.standard.             var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.ff.motorCharacterization.var = Simulink.Variant( 'mdl_case == 1' );

io.write.serial.hardware.pid.standard.            var = Simulink.Variant( 'mdl_case == 0' );
io.write.serial.hardware.pid.sensorCalibration.   var = Simulink.Variant( 'mdl_case == 2' );

% plant: nonlinearDynamics

io.write.serial.nonlinearDynamics.                var = Simulink.Variant( 'plant_dynamics_mode == 1' );

io.write.serial.nonlinearDynamics.ff.             var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
io.write.serial.nonlinearDynamics.pid.            var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );

io.write.serial.nonlinearDynamics.ff.standard.      var = Simulink.Variant( 'mdl_case == 0' );

io.write.serial.nonlinearDynamics.pid.standard.     var = Simulink.Variant( 'mdl_case == 0' );

% plant: nonlinearDynamics

io.write.serial.linearDynamics.                     var = Simulink.Variant( 'plant_dynamics_mode == 2' );

io.write.serial.linearDynamics.ff.                  var = Simulink.Variant( 'ctrl_motor_v_mode == 0' );
io.write.serial.linearDynamics.pid.                 var = Simulink.Variant( 'ctrl_motor_v_mode == 1' );

io.write.serial.linearDynamics.ff.standard.         var = Simulink.Variant( 'mdl_case == 0' );

io.write.serial.linearDynamics.pid.standard.        var = Simulink.Variant( 'mdl_case == 0' );

%% [Init    ]: Write commands
io.write.ctrl.motor_v.cmd.tStart          = ui.io.write.ctrl.motor_v.cmd.tStart;          % [   s           ]
io.write.ctrl.motor_v.cmd.val.x           = ui.io.write.ctrl.motor_v.cmd.val.x;           % [ <cmd>         ]
io.write.ctrl.motor_v.cmd.val_norm.dx.max = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.max; % [  cmd.norm / s ]
io.write.ctrl.motor_v.cmd.val_norm.dx.min = ui.io.write.ctrl.motor_v.cmd.val_norm.dx.min; % [  cmd.norm / s ]

%% End
```

Code Listing 1.11: [minseg.m]: Initialization - Model - User-Defined Board Inputs and Outputs

1.1.3.2.5 Build Parameters

Code Listing 1.12: [minseg.m]: Initialization - Model - Model Build Parameters

```
%% [Init    ]: Initialize list of general parameters used within Simulink model

mdl.parameter.label = {};

% Specify parameters which will be used in model:
mdl.parameter.label = [...
mdl.parameter.label
{
  'k.intmax.uint8'

  'mdl.mode'
  'mdl.case'
  'mdl.T.sample'

  'plant.dynamics.mode'
  'plant.supply.v'

  'ctrl.motor_v.mode'
  'ctrl.motor_v.ff.motor_v.var'
  'ctrl.motor_v.pid.motor_w.var'

  'io.write.serial.var'
  'io.write.scopes.var'

  'io.write.serial.hardware.var'
  'io.write.serial.hardware.ff.var'
  'io.write.serial.hardware.ff.standard.var'
  'io.write.serial.hardware.ff.motorCharacterization.var'
  'io.write.serial.hardware.pid.var'
  'io.write.serial.hardware.pid.standard.var'
  'io.write.serial.hardware.pid.sensorCalibration.var'

  'io.write.serial.nonlinearDynamics.var'
  'io.write.serial.nonlinearDynamics.ff.var'
  'io.write.serial.nonlinearDynamics.ff.standard.var'
  'io.write.serial.nonlinearDynamics.pid.var'
  'io.write.serial.nonlinearDynamics.pid.standard.var'

  'io.write.serial.linearDynamics.var'
  'io.write.serial.linearDynamics.ff.var'
  'io.write.serial.linearDynamics.ff.standard.var'
  'io.write.serial.linearDynamics.pid.var'
  'io.write.serial.linearDynamics.pid.standard.var'

  'io.write.ctrl.motor_v.cmd.tStart'
  'io.write.ctrl.motor_v.cmd.val.x'
  'io.write.ctrl.motor_v.cmd.val_norm.dx.max'
  'io.write.ctrl.motor_v.cmd.val_norm.dx.min'

}];

%% [Init    ]: Append case-dependent parameters: Plant: Dynamics model

switch plant.dynamics.mode

case 0 % hardware
mdl.parameter.label = [...
mdl.parameter.label
{
  'gyro.dlpf.mode'
  'gyro.k_raw2actual'

  'gyro.x.bias'
  'gyro.x.reset'
  'gyro.y.bias'
  'gyro.y.reset'
  'gyro.z.bias'
  'gyro.z.reset'

  'gyro.filter.z.ss.A'
  'gyro.filter.z.ss.B'
  'gyro.filter.z.ss.C'
  'gyro.filter.z.ss.D'

  'gyro.filter.z.num'
  'gyro.filter.z.den'

  'accel.k_raw2actual'

  'mtr.driver.left.pin.pos'
  'mtr.driver.left.pin.neg'
  'mtr.driver.middle.pin.pos'
  'mtr.driver.middle.pin.neg'

  'mtr.encoder.left.pin.A'
  'mtr.encoder.left.pin.B'
  'mtr.encoder.middle.pin.A'
  'mtr.encoder.middle.pin.B'

  'mtr.encoder.countPerRev'
  'mtr.encoder.radPerRev'

  'mtr.encoder.filter.z.ss.A'
  'mtr.encoder.filter.z.ss.B'
  'mtr.encoder.filter.z.ss.C'
  'mtr.encoder.filter.z.ss.D'

  'mtr.encoder.filter.z.num'
  'mtr.encoder.filter.z.den'
}];

case 1
mdl.parameter.label = [...
mdl.parameter.label
{
}];

case 2
mdl.parameter.label = [...
mdl.parameter.label
{
}];

end

%% [Init    ]: Append case-dependent parameters: Controller: v.motor.input

switch ctrl.motor_v.mode

case 0 % feed-forward (input: motor.v)
```

```
mdl.parameter.label = [...
mdl.parameter.label
{

}];

case 1 % PID           (input: motor.w)
mdl.parameter.label = [...
mdl.parameter.label
{
   'ctrl.motor_v.pid.motor_w.k.p'
   'ctrl.motor_v.pid.motor_w.k.i'
   'ctrl.motor_v.pid.motor_w.k.d'

   'ctrl.motor_v.pid.motor_w.int.maxVal'
   'ctrl.motor_v.pid.motor_w.int.minVal'
}];

end

%% [Init    ]: Relabel parameters for use within Simulink model

% Number of parameters specified
mdl.n.parameter         = size( mdl.parameter.label, 1);

% Indices which contain periods:
mdl.parameter.z.period = regexp(mdl.parameter.label, '\.');

% For each parameter:
for i0 = 1 : mdl.n.parameter

% Create a new label in which all periods have been set to underscores:
mdl.parameter.label0 = mdl.parameter.label    {i0,1};
mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';

% Set the data for the new label equal to the data from the old label:
eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';' ]);

end

%% [Init    ]: Refresh model to update variant blocks
```

```

mdl.object.refreshModelBlocks

%% End
```

Code Listing 1.12: [minseg.m]: Initialization - Model - Model Build Parameters

1.1.3.3 Serial

1.1.3.3.1 Write

Code Listing 1.13: [minseg.m]: Initialization - Serial - Write

```

%% End
```

Code Listing 1.13: [minseg.m]: Initialization - Serial - Write

1.1.3.3.2 Read

Code Listing 1.14: [minseg.m]: Initialization - Serial - Read

```
%% [Init    ]: Import serial read signal label and datatype from model

% serial read block location:
srl.read{1,1}.block.path = ...
  [  mdl.label '/Board Input // Output/Writes (To PC)/Serial' ];

while 1 % continue until 'break' command

  srl.read{1,1}.block.path0 = get_param(  srl.read{1,1}.block.path         ...
                                        , 'ActiveVariantBlock'              ...
                                        );

  if isempty( srl.read{1,1}.block.path0 )
  break
  end

  srl.read{1,1}.block.path  = srl.read{1,1}.block.path0;

end

% serial read block names:
srl.read{1,1}.block.busSelect.label =                        ...
find_system(  srl.read{1,1}.block.path                       ...
           , 'Regexp', 'on'                                  ...
           , 'Name',   'Bus Selector'                        ...
           );

srl.read{1,1}.block.convert.label =                          ...
find_system(  srl.read{1,1}.block.path                       ...
           , 'Regexp', 'on'                                  ...
           , 'Name',   'Data Type Conversion*'               ...
           );

srl.read{1,1}.block.bytepack.label =                         ...
find_system(  srl.read{1,1}.block.path                       ...
           ,  'Regexp', 'on'                                 ...
           ,  'Name',   'Byte Pack*'                         ...
           );
```

```

% import output signal labels from bus block
srl.read{1,1}.block.busSelect.signals.out =               ...
get_param  ( srl.read{1,1}.block.busSelect.label          ...
           , 'OutputSignals'                              ...
           );

srl.read{1,1}.block.busSelect.signals.out =               ...
regexp     ( srl.read{1,1}.block.busSelect.signals.out{:} ...
           , '[^,]*'                                      ...
           , 'match'                                      ...
           ).';

% verify equivalent number of each type of serial read preprocessing block:
assert(    size( srl.read{1,1}.block.busSelect.signals.out, 1 ) == ...
           size( srl.read{1,1}.block.convert.label,         1 )    ...
      , [ srl.read{1,1}.block.path ':\n'                           ...
         'Less Convert blocks than number of signals.']            ...
      )

assert(    size( srl.read{1,1}.block.busSelect.signals.out, 1 ) == ...
           size( srl.read{1,1}.block.bytepack.label, 1 )           ...
      , [ srl.read{1,1}.block.path ':\n'                           ...
         'Less Byte Pack blocks than number of signals.']          ...
      )

%% [Init   ]: Define serial read signal label and datatype parameters

% number of signals being transmitted:
srl.read {1,1}.n.signals = size( srl.read{1,1}.block.convert.label, 1 );

% increase srl.read cell vector size to number of signals
srl.read { srl.read{1,1}.n.signals, 1 } = [];
srl.reads{ srl.read{1,1}.n.signals, 1 } = [];


% for each serial read signal existing within the model:
for i0 = 1 : srl.read{1,1}.n.signals

  % import the datalabel of that signal from the bus block
  srl.read{i0,1}.label = srl.read{1,1}.block.busSelect.signals.out{i0,1};

  % import the datatype of that signal from the datatype conversion block
  srl.read{i0,1}.type.original = ...
  get_param( srl.read{ 1,1}.block.convert.label{i0,1}, 'OutDataTypeStr' );

  % for posterity, set the datatype in the bytepack block to the same datatype.
  set_param( srl.read{ 1,1}.block.bytepack.label{i0,1}, 'datatypes',      ...
      ['{''' srl.read{i0,1}.type.original '''}' ]                          );

end

%% [Init   ]: Define serial read signal size parameters

% initialize counters
srl.read{1,1}.n.Bytes       = 0; % [bytes                  / read]

srl.read{1,1}.n.type.uint8  = 0; % [type: 'uint8'  signals / read]
srl.read{1,1}.n.type.uint16 = 0; % [type: 'uint16' signals / read]
srl.read{1,1}.n.type.uint32 = 0; % [type: 'uint32' signals / read]

srl.read{1,1}.n.type.int8   = 0; % [type: 'int8'   signals / read]
srl.read{1,1}.n.type.int16  = 0; % [type: 'int16'  signals / read]
srl.read{1,1}.n.type.int32  = 0; % [type: 'int32'  signals / read]

srl.read{1,1}.n.type.single = 0; % [type: 'single' signals / read]
srl.read{1,1}.n.type.double = 0; % [type: 'double' signals / read]

for i0 = 1 : srl.read{1,1}.n.signals

  % increment counter for appropriate signal type [ - ]
  switch srl.read{i0,1}.type.original
  case 'uint8' ; srl.read{1,1}.n.type.uint8  = srl.read{1,1}.n.type.uint8  + 1;
  case 'uint16'; srl.read{1,1}.n.type.uint16 = srl.read{1,1}.n.type.uint16 + 1;
  case 'uint32'; srl.read{1,1}.n.type.uint32 = srl.read{1,1}.n.type.uint32 + 1;

  case 'int8'  ; srl.read{1,1}.n.type.int8   = srl.read{1,1}.n.type.int8   + 1;
  case 'int16' ; srl.read{1,1}.n.type.int16  = srl.read{1,1}.n.type.int16  + 1;
  case 'int32' ; srl.read{1,1}.n.type.int32  = srl.read{1,1}.n.type.int32  + 1;

  case 'single'; srl.read{1,1}.n.type.single = srl.read{1,1}.n.type.single + 1;
  case 'double'; srl.read{1,1}.n.type.double = srl.read{1,1}.n.type.double + 1;
  otherwise;      error('unknown datatype');
  end

  switch srl.read{i0,1}.type.original
  case 'uint8' ; srl.read{i0,1}.n.bytes = 1;       % [ (bytes/signal) / read ]
  case 'uint16'; srl.read{i0,1}.n.bytes = 2;       % [ (bytes/signal) / read ]
  case 'uint32'; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]

  case 'int8'  ; srl.read{i0,1}.n.bytes = 1;       % [ (bytes/signal) / read ]
  case 'int16' ; srl.read{i0,1}.n.bytes = 2;       % [ (bytes/signal) / read ]
  case 'int32' ; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]

  case 'single'; srl.read{i0,1}.n.bytes = 4;       % [ (bytes/signal) / read ]
  case 'double'; srl.read{i0,1}.n.bytes = 8;       % [ (bytes/signal) / read ]
  otherwise;      error('unknown datatype');
  end

  srl.read{i0,1}.n.bits  = srl.read{i0,1}.n.bytes  ...
                         * k.byte2bit;             % [ (bits /signal) / read ]

  srl.read{1 ,1}.n.Bytes = srl.read{1 ,1}.n.Bytes  ...
                         + srl.read{i0,1}.n.bytes; % [  bytes          / read ]

end

  srl.read{1 ,1}. n.Bits = srl.read{1 ,1}.n.Bytes  ...
                         * k.byte2bit;             % [  bits           / read ]

% verify number of bytes per read is not greater than arduino input buffer:
assert( (srl.read{1,1}.n.Bytes + 1) <= 64                 ...
      , ['Number of bytes being sent per read '         ...
         '(including 1 byte for Terminator)\n'          ...
         'is greater than size of\n'                    ...
         'Arduino Mega 2650 input buffer (64 bytes).'] ...
      )

%% [Init   ]: Initialize serial read value vectors

for i0 = 1 : srl.read{1,1}.n.signals
srl.read {i0,1}.val = zeros( srl.read {i0,1}.n.bytes, 1 ); % [varies]
end
```

```

srl.read {1 ,1}.Val = zeros( srl.read {1 ,1}.n.Bytes, 1 ); % [varies]

%% End
```

Code Listing 1.14: [minseg.m]: Initialization - Serial - Read

1.1.3.3.3 General

Code Listing 1.15: [minseg.m]: Initialization - Serial - General

```
%% [Init    ]: Define serial communication parameters (general)

% serial address on PC
switch ui.srl.mode.address
  case 0; srl.address = '/dev/tty.usbmodem1411'; % left       usb port (2015 PC)
  case 1; srl.address = '/dev/tty.usbmodem621';  % left-rear usb port (2008 PC)
end
% note: to determine current address, use command: {ls /dev/tty.*} in Terminal.app

srl.byteOrder          = 'littleEndian';       % [-]
srl.f.baud             =  115200;              % [bit /   s]
srl.T.baud             =  1 / srl.f.baud;      % [  s / bit]

srl.type.in            = 'uint8'; % signal datatype when entering transmission
srl.type.out           = 'uint8'; % signal datatype when exiting  transmission

% legend:
% read  involves a single read  (1   sample ).
% reads involves      all reads (all samples).

%% [Init    ]: Serial buffer size

srl.bufferSize.in  = max( [0; srl.read{1,1}.n.Bits] ); % [bits]
srl.bufferSize.out = srl.bufferSize.in;                % [bits]

% buffer sizes should be equivalent to write or read size (whichever is higher).

%% [Process]: Setup serial object

% Ensure that desired serial port does not already exist in the loaded list:
if ~isempty( instrfind('Port', srl.address) )
  fclose   ( instrfind('Port', srl.address) );
  delete   ( instrfind('Port', srl.address) );
end

% Initialize serial object
srl.srl = serial(                     srl.address        ...
                 , 'ByteOrder'       , srl.byteOrder      ...
```

```
                , 'BaudRate'         , srl.f.baud          ... [ Hz ]
                , 'InputBufferSize'  , srl.bufferSize.in   ... [bits]
                , 'OutputBufferSize', srl.bufferSize.out ... [bits]
                );

% For detailed information, use: get(srl.srl)

%{
how prove no "header" value?
how read timeout period? how reduce to something reasonable?

find more information on:

  TimerPeriod = 1
  Timeout     = 10
  StopBits    = 1

%}

%% [Init   ]: Time required to perform transmission

% time required to transmit each write:
srl.write{1,1}.T.transmit = srl.T.baud * (            0            ); % [ s / write ]
                         %  [s / bit] * (       [bit / write]       )

% time required to transmit each read:
srl.read {1,1}.T.transmit = srl.T.baud * ( srl.read{1,1}.n.Bits + 08 ); % [ s / read ]
                         %  [s / bit] * (         [bit / read]        )
% note: 1 byte (08 bits) added to account for terminator (1 byte).

srl.read {1,1}.T.transmit = srl.read {1,1}.T.transmit * 10 / 08;

% time required to perform all transmissions:
srl.T.transmit = srl.write{1,1}.T.transmit + srl.read{1,1}.T.transmit;  % [ s ]

%% [Init   ]: Time between start of each transmission

% number of board sample periods per serial process period

```

```
if ui.srl.T.decimation == 0
srl.T.decimation = ceil( srl.T.transmit * 1.0000 / mdl.T.sample );

else
srl.T.decimation = ui.srl.T.decimation;

end

% time until next serial process:
srl.T.sample = mdl.T.sample * srl.T.decimation;                          % [ s ]

%% [Init   ]: Verify serial period

% verify that total time to transmit serial data is not greater than
%  time until start of next serial process:
assert( srl.T.transmit < srl.T.sample                    ...
      , 'Read period is greater than sample period.' ...
      );

%% [Init   ]: Define serial transmits parameters

% number of reads to perform:
% note: serial duration may be specified directly in terms of samples or in terms of time
try     srl.n.transmits = round( ui.srl.T.transmits / srl.T.sample ); % [transmit cycles]
catch; srl.n.transmits =         ui.srl.n.transmits;                  % [transmit cycles]
end

srl.n.transmits = srl.n.transmits + 1; % 1 added for time = 0

%% End
```

Code Listing 1.15: [minseg.m]: Initialization - Serial - General

1.1.3.3.4 Reads

Code Listing 1.16: [minseg.m]: Initialization - Serial - Reads

```
%% [Init    ]: Initialize serial reads variable

srl.reads{ srl.read{1,1}.n.signals , 1 } = [];



%% [Init    ]: Define serial reads parameters

% number of bytes/bits captured after all reads have been performed:
for i0 = 1 : srl.read{1,1}.n.signals
srl.reads{i0,1}.n.bytes = srl.read{i0,1}.n.bytes * srl.n.transmits; % [bytes]
srl.reads{i0,1}.n.bits  = srl.read{i0,1}.n.bits  * srl.n.transmits; % [bits ]
end

srl.reads{1 ,1}.n.Bytes = srl.read{1 ,1}.n.Bytes * srl.n.transmits; % [bytes]
srl.reads{1 ,1}.n.Bits  = srl.read{1 ,1}.n.Bits  * srl.n.transmits; % [bits ]

%% [Init    ]: Initialize serial reads value vectors

for i0 = 1 : srl.read{1,1}.n.signals
srl.reads{i0,1}.val = zeros( srl.reads{i0,1}.n.bytes, 1 ); % [varies]
end

srl.reads{1 ,1}.Val = zeros( srl.reads{1 ,1}.n.Bytes, 1 ); % [varies]

%% End
```

Code Listing 1.16: [minseg.m]: Initialization - Serial - Reads

1.1.3.3.5 Build Parameters

Code Listing 1.17: [minseg.m]: Initialization - Serial - Model Build Parameters

```
%% [Init    ]: Define serial transmission parameters
io.srl.read.rateTransition.initialCondition = uint8( zeros( srl.read{1,1}.n.Bytes, 1 ) );
io.srl.read.rateTransition.T.sample          = srl.T.sample;

%% [Init    ]: Initialize list of general parameters used within Simulink model

mdl.parameter.label = {};

% Specify parameters which will be used in model:
mdl.parameter.label = [...
mdl.parameter.label
{
  'io.srl.read.rateTransition.initialCondition'
  'io.srl.read.rateTransition.T.sample'

%  cannot set certain hardware parameters via variables. [must hard-code.]

% 'srl.address';
% 'srl.f.baud';
}];

%% [Init    ]: Relabel parameters for use within Simulink model

% Number of parameters specified
mdl.n.parameter          = size( mdl.parameter.label, 1);

% Indices which contain periods:
mdl.parameter.z.period = regexp(mdl.parameter.label, '\.');

% For each parameter:
for i0 = 1 : mdl.n.parameter

% Create a new label in which all periods have been set to underscores:
mdl.parameter.label0 = mdl.parameter.label   {i0,1};
mdl.parameter.label0 ( mdl.parameter.z.period{i0,1} ) = '_';

% Set the data for the new label equal to the data from the old label:
```

```
eval([ mdl.parameter.label0 ' = ' mdl.parameter.label{i0,1} ';' ]);

end

%% [Init    ]: Update model scan for errors

set_param(mdl.label, 'SimulationCommand', 'update' )

%% End
```

Code Listing 1.17: [minseg.m]: Initialization - Serial - Model Build Parameters

1.1.4 Processing

1.1.4.1 Build

Code Listing 1.18: [minseg.m]: Processing - Build

```
%% [Process]: Build (Normal mode or External mode)
  switch mdl.mode

  case 0 % Normal mode
  disp('Performing build:')
  mdl.T.build    = tic;
  set_param(mdl.label, 'SimulationMode',    'normal'  ) % put model into normal mode
  rtwbuild (mdl.label                                 ) % build model into hardware
  disp('Build completed.')
  disp(' ')

  case 1 % External mode
  set_param(mdl.label, 'SimulationMode',    'external') % put model into external mode
  set_param(mdl.label, 'SimulationCommand', 'connect' ) % connect to the executable
  set_param(mdl.label, 'SimulationCommand', 'start'   ) % start      the executable
% set_param(mdl.label, 'SimulationCommand', 'stop'    ) % stop       the executable

  end

%% End
```

Code Listing 1.18: [minseg.m]: Processing - Build

1.1.4.2 Serial Transmission

Code Listing 1.19: [minseg.m]: Processing - Serial - Transmit

```
%% [Process]: Open, read/write, and close serial port object.

% open serial channel
fopen ( srl.srl );

disp( 'Performing serial read:' )

% initialize complete read cycle timers
srl.t.start = clock;
srl.T.all   = tic;

for i0 = 1 : srl.n.transmits
srl.            T.one = tic;


% write
% srl.write{1,1}.T.one = tic;


% read
srl.read{1,1}.T.one = tic;


% perform read of one time sample:
srl.read{1,1}.Val = fread( srl.srl                  ... serial object
                         , srl.read{1,1}.n.Bytes   ... read size        [bytes/read]
                         , srl.type.in             ... input data class [default: 'uint8']
                         );

if isempty( srl.read{1,1}.Val ) % occasionally isempty on startup.    [seek better fix.]
srl.read{1,1}.Val = NaN * zeros( srl.read {1 ,1}.n.Bytes, 1 );
end

% append to vector of all reads:
srl.reads{1,1}.Val( ( 1:srl.read{1,1}.n.Bytes ) + (i0-1)*srl.read{1,1}.n.Bytes, 1) = ...
srl.read {1,1}.Val;


% wait for end of time sample:
if i0 ~= srl.n.transmits                 % if not the last sample
  while toc( srl.T.one ) < srl.T.sample  % then loop to wait until
  end                                    % a complete sample period
end                                      % has passed before reading
                                         % again.

end

srl.T.all  = toc( srl.T.all );
srl.t.stop = clock;

disp(['Intended total transmit time: ' num2str( srl.n.transmits * srl.T.sample, '%010.6f' ) ]);
disp(['Actual   total transmit time: ' num2str( srl.T.all                     , '%010.6f' ) ]);
disp( 'Serial read complete.' )
disp( ' ' )


fclose( srl.srl );


% convert output to intended data type:
srl.read {1,1}.Val = cast( srl.read {1,1}.Val, srl.type.out );
srl.reads{1,1}.Val = cast( srl.reads{1,1}.Val, srl.type.out );
% note: Mathworks forces conversion to 'double' for serial read output.

%% End
```

Code Listing 1.19: [minseg.m]: Processing - Serial - Transmit

1.1.4.3 Serial Reads Post-Processing

Code Listing 1.20: [minseg.m]: Processing - Serial - Reads

```
%% [Process]: Format serial port data

% Index of first byte of each read
srl.reads{1,1}.z.byte1 = ( 0:srl.n.transmits-1 ).' * srl.read{1,1}.n.Bytes + 1;

srl.read {1,1}.i.byte0 = 0; % initialize byte offset
for i0 = 1 : srl.read{1,1}.n.signals

  % Start index of signal i0 at each sample
  srl.reads{i0,1}.z.byte0   = srl.reads{1,1}.z.byte1 + srl.read{1,1}.i.byte0;

  % Include additional indices for multibyte signals
  if srl.read {i0,1}.n.bytes > 1
     srl.reads{i0,1}.z.byte0 = bsxfun( @plus                            ...
                                     ,   srl.reads{i0,1}.z.byte0   ...
                                     , 0:srl.read {i0,1}.n.bytes-1 ...
                                     );
  end

  % Pull corresponding values
  if strcmp( srl.read{i0,1}.type.original, srl.type.out )
       % If intended signal datatype is     equal to serial output,
       % then use it immediately:
    srl.reads{i0,1}.val   = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );

  else % If intended signal datatype is not equal to serial output,
       % then first convert the serial output:
    srl.reads{i0,1}.val0 = srl.reads{1,1}.Val( srl.reads{i0,1}.z.byte0 );

    % Convert to cell:
    srl.reads{i0,1}.val   = mat2cell(             srl.reads{i0,1}.val0          ...
                                    , ones( size( srl.reads{i0,1}.val0, 1 ), 1) ...
                                    ,       size( srl.reads{i0,1}.val0, 2 )     ...
                                    );
    % Typecast each row vector to correct type:
    srl.reads{i0,1}.fun   = @(x) typecast( x, srl.read{i0,1}.type.original );
    srl.reads{i0,1}.val   = cellfun ( srl.reads{i0,1}.fun                       ...
                                    , srl.reads{i0,1}.val                       ...
```

```
                                       );
    % Convert back to matrix: [unnecessary - cellfun converts to matrix already]
    % srl.read{i0,1}.val  = cell2mat( srl.read{i0,1}.val );

    % Determine maximum and minumum values (axis information in plots)
    srl.reads{i0,1}.val_min    = min( srl.reads{i0,1}.val );
    srl.reads{i0,1}.val_max    = max( srl.reads{i0,1}.val );
    srl.reads{i0,1}.val_absMax = max( abs( [ srl.reads{i0,1}.val_min
                                             srl.reads{i0,1}.val_max ] ) );
  end



  % Increment byte offset
  srl.read{1,1}.i.byte0 = srl.read{1,1}.i.byte0 + srl.read{i0,1}.n.bytes;

end

%% End
```

Code Listing 1.20: [minseg.m]: Processing - Serial - Reads

1.1.5 Output

1.1.5.1 Save

Code Listing 1.21: [minseg.m]: Output - Save

```matlab
%% [Output]: Save all data

file.label = [datestr(now, 'yyyy.mm.dd HH.MM') ' minseg'];

if ~isempty(ui.save.label)
file.label = [ file.label ' ' ui.save.label ];
end

disp( 'Performing export to .mat file.' )

save( [root.data.dir file.label '.mat' ] )

disp( 'Export to .mat file complete.'   )
disp( ' '                               )

%% End
```

Code Listing 1.21: [minseg.m]: Output - Save

1.1.5.2 Serial Reads Plot

Code Listing 1.22: [minseg.m]: Output - Serial - Reads - Plot

```
%% [Output ]: Common plot commands

%subplot with 2d indices:
dim1     = @(n_col, row, col)         (row-1)*n_col + col; % Matrix index: 2d to 1d
subplott = @(n_row, n_col, M) subplot(n_row, n_col, dim1(n_col, M(1), M(2)) );

% axis value
msd      = @(x)     fix( log10( abs(x) ) );                          % most significant digit. [ones digit = 0th digit]
rndout   = @(x, N) sign(x) .* ceil( abs(x)*10^(-N) ) * 10^(+N); % round away from zero at specified digit.
rndOut   = @(x, N) rndout(x, msd(x) - N ); % round away from zero at N digits right from most significant digit.

%% [Output ]: Plot setup
p = 0;

disp( 'Performing plot creation:' )

for i0 = 1 : srl.read{1,1}.n.signals

  p = p + 1;
  figure(p)

  % plot data
  if i0==1; stairs(                        srl.reads{i0,1}.val, '.-'); % clock
  else;     stairs(srl.reads{1,1}.val, srl.reads{i0,1}.val, '.-'); % all else
  end

  % labels
  if i0==1; xlabel( 'Samples [-]' ); % clock
  else;     xlabel( 'Time [s]'    ); % all else
  end

  ylabel( srl.read{i0,1}.label )

  % y-axis limits
  if isa(srl.reads{i0,1}.val, 'float') % float
```

```
    srl.reads{i0,1}.ymin = -rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
    srl.reads{i0,1}.ymax = +rndOut(srl.reads{i0,1}.val_absMax+eps, 2);
  else                                  % integer
    srl.reads{i0,1}.ymin = double( intmin( class(srl.reads{i0,1}.val) ) );
    srl.reads{i0,1}.ymax = double( intmax( class(srl.reads{i0,1}.val) ) );
  end

  ylim([ srl.reads{i0,1}.ymin, srl.reads{i0,1}.ymax ])

  grid minor

end

disp( 'Plot creation complete.'   )
disp( ' '                         )


%% [Output ]: Close legacy figures

% not yet implemented.
% use "a = get(groot, 'Children')" to list all figures.
% then "for all figures: if n.figure > n.figure.gcf, close n.figure"

% when implemented, stop using "close all"

%% End
```

Code Listing 1.22: [minseg.m]: Output - Serial - Reads - Plot

1.1.6 Global Cleanup

Code Listing 1.23: [minseg.m]: Global Cleanup

```
%% [Cleanup]: Remove alternate subdirectories from Matlab path

Simulink.fileGenControl('reset')

%% [Cleanup]: Remove alternate subdirectories from Simulink path

for i0 = 1 : root.n.sub.dir
  rmpath(    root.   sub.dir{ root.n.sub.dir - (i0 - 1), 1 } )
end

%% End
```

Code Listing 1.23: [minseg.m]: Global Cleanup