

What is the best way to implement my algorithm in Simulink® ?

**By Giampiero Campa, PhD, Technical Evangelist
MathWorks, 970 W 190 ST, Suite 530,
Torrance, CA, 90502, USA**

giampiero.campa@mathworks.com

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

An overview of options

- MATLAB® based:
 - MATLAB S-functions
 - MATLAB functions
 - MATLAB System objects™

- C based:
 - C S-functions
 - S-Function Builder
 - Legacy Code Tool (LCT)

- Simulink based:
 - Assembling Simulink blocks

Automatic code (and executable) generation

- No code generation allowed:
 - MATLAB S-functions

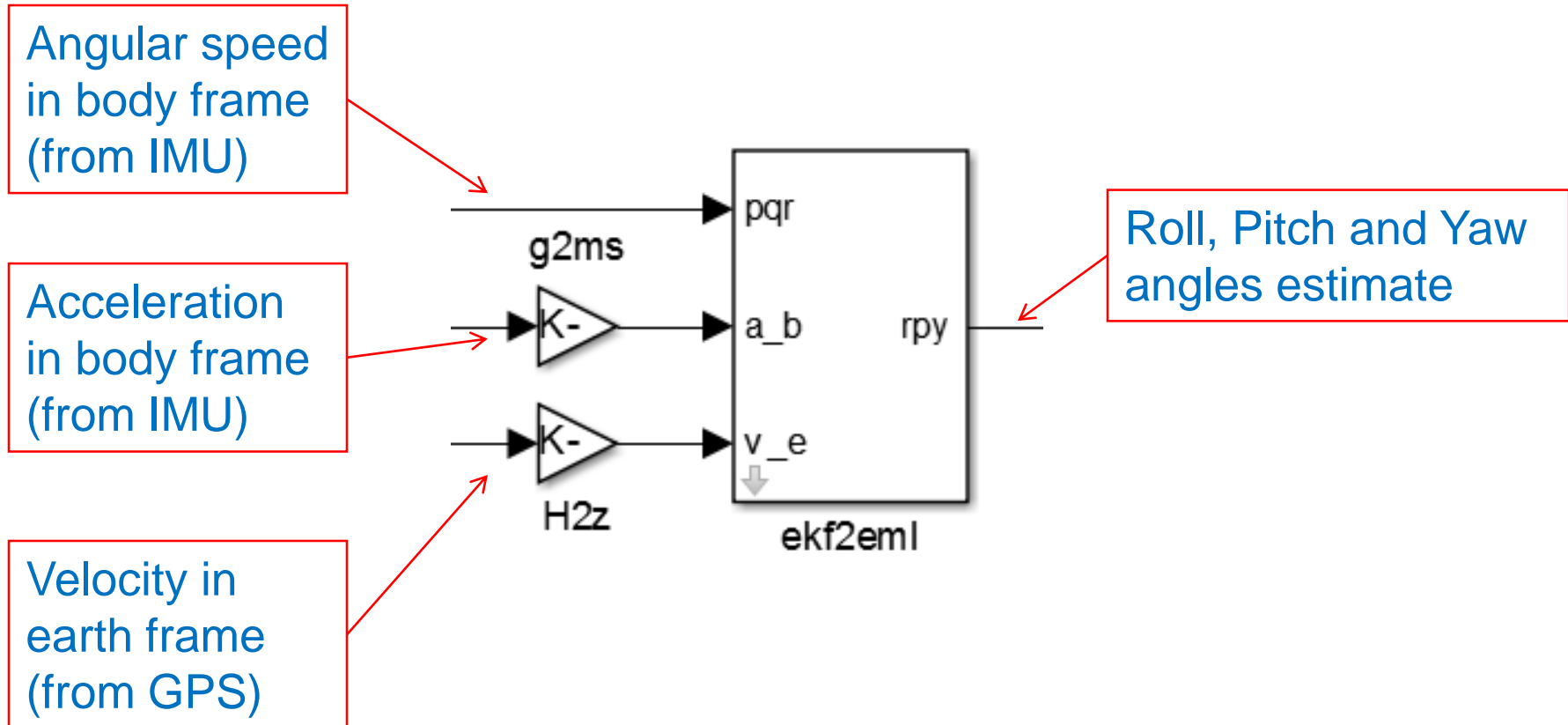
- Only toward targets supporting noninlined S-functions:
 - C S-functions

- Code generation allowed toward any target:
 - MATLAB functions
 - MATLAB System Objects
 - Legacy Code Tool
 - S-Function Builder
 - Assembling Simulink blocks

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

An EKF-based GPS/IMU sensor fusion algorithm for attitude estimation



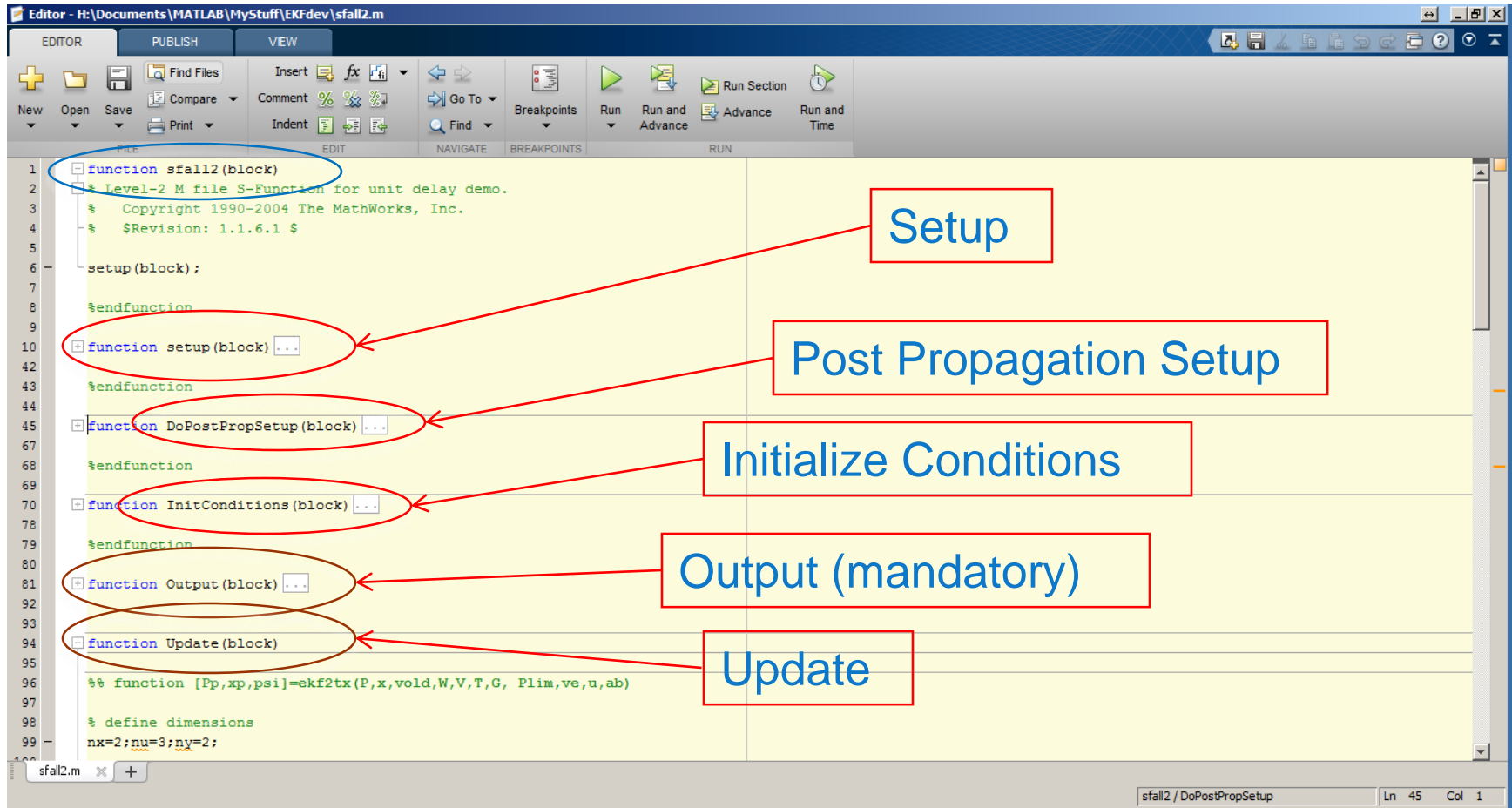
EKF for GPS/IMU sensor fusion: summary

- 3 inputs, each one of size 3×1
- 1 output, also having size 3×1
- Using simplified solution relying only on internal roll and pitch estimates (Kingston-Beard)
 - Internal states are: roll and pitch estimates, a 2×2 P matrix, and the previous velocity in body frame (3×1)
 - Only minor linear algebra required (few 2×2 matrix multiplications and one inversion), so manual coding in C is affordable
- So how do we implement this ?

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- **Case Study: Implementing the EKF as a Simulink block**
- Informal performance comparison
- Conclusions

MATLAB S-function block (level 2)



Editor - H:\Documents\MATLAB\MyStuff\EKFdev\sfall2.m

EDITOR PUBLISH VIEW

FILE EDIT NAVIGATE BREAKPOINTS RUN

1 **function** sfall2(block)

2 % Level-2 M file S-Function for unit delay demo.

3 % Copyright 1990-2004 The MathWorks, Inc.

4 % \$Revision: 1.1.6.1 \$

5

6 setup(block);

7

8 **endfunction**

9

10 **function** setup(block) ...

11

12 **endfunction**

13

14 **function** DoPostPropSetup(block) ...

15

16 **endfunction**

17

18 **function** InitConditions(block) ...

19

20 **endfunction**

21

22 **function** Output(block) ...

23

24 **function** Update(block)

25

26 %% function [Pp,xp,psi]=ekf2tx(P,x,vold,W,V,T,G, Plim,ve,u,ab)

27

28 % define dimensions

29 nx=2;nu=3;ny=2;

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

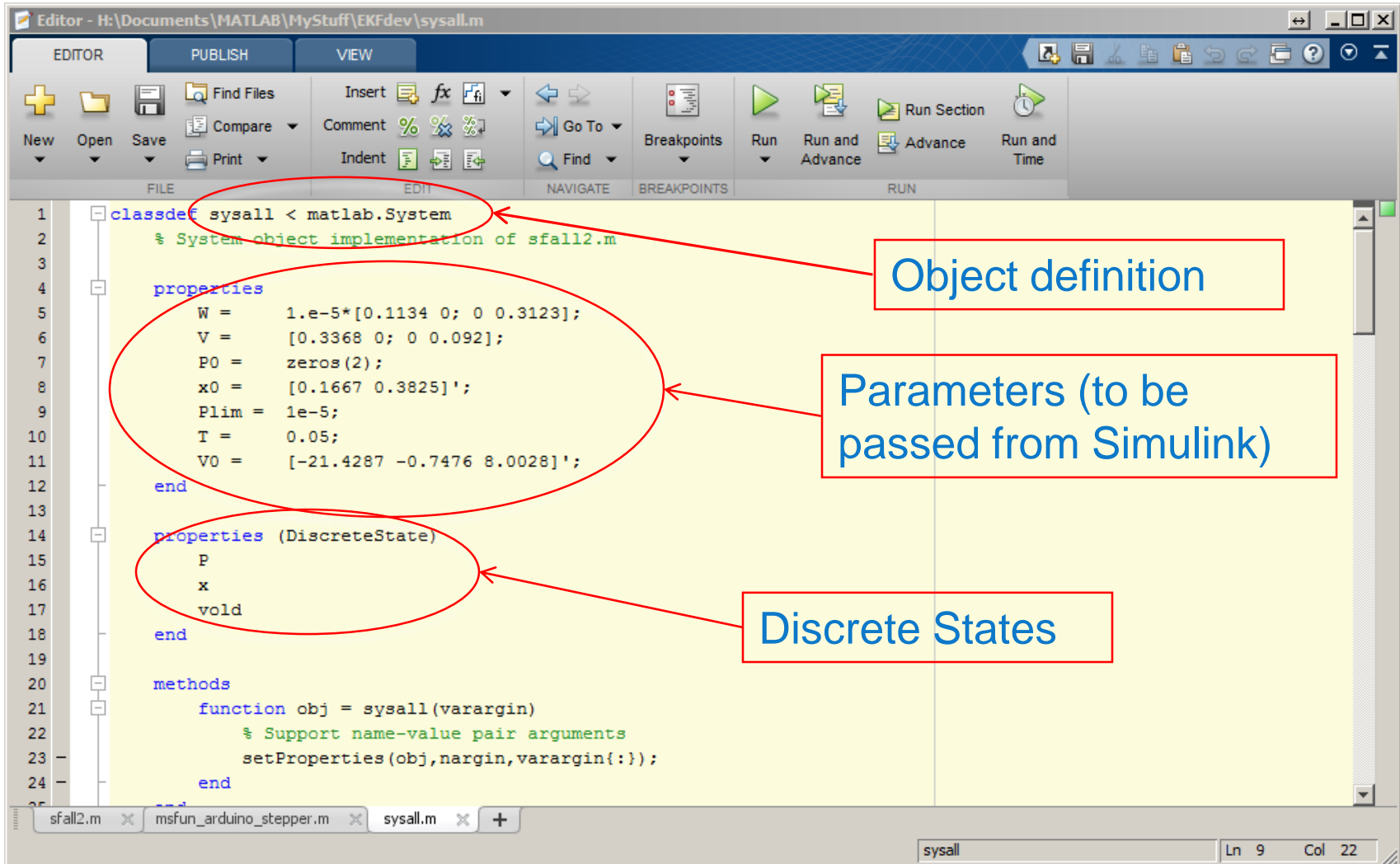
1300

1301

MATLAB S-function block: pros and cons

- Allows **fine control** of sizes, inputs, outputs, states, work vectors, etc.
- Allows use of any MATLAB function, toolbox, or data structure (with few limitations).
- Is **interpreted** (may be slower).
- Does **not allow code generation** and targeting (may only be used for simulation).

MATLAB System object block



```

1 classdef sysall < matlab.System
2     % System object implementation of sfall2.m
3
4     properties
5         W = 1.e-5*[0.1134 0; 0 0.3123];
6         V = [0.3368 0; 0 0.092];
7         P0 = zeros(2);
8         x0 = [0.1667 0.3825]';
9         Plim = 1e-5;
10        T = 0.05;
11        V0 = [-21.4287 -0.7476 8.0028]';
12    end
13
14    properties (DiscreteState)
15        P
16        x
17        vold
18    end
19
20    methods
21        function obj = sysall(varargin)
22            % Support name-value pair arguments
23            setProperties(obj,nargin,varargin{:});
24        end
25    end

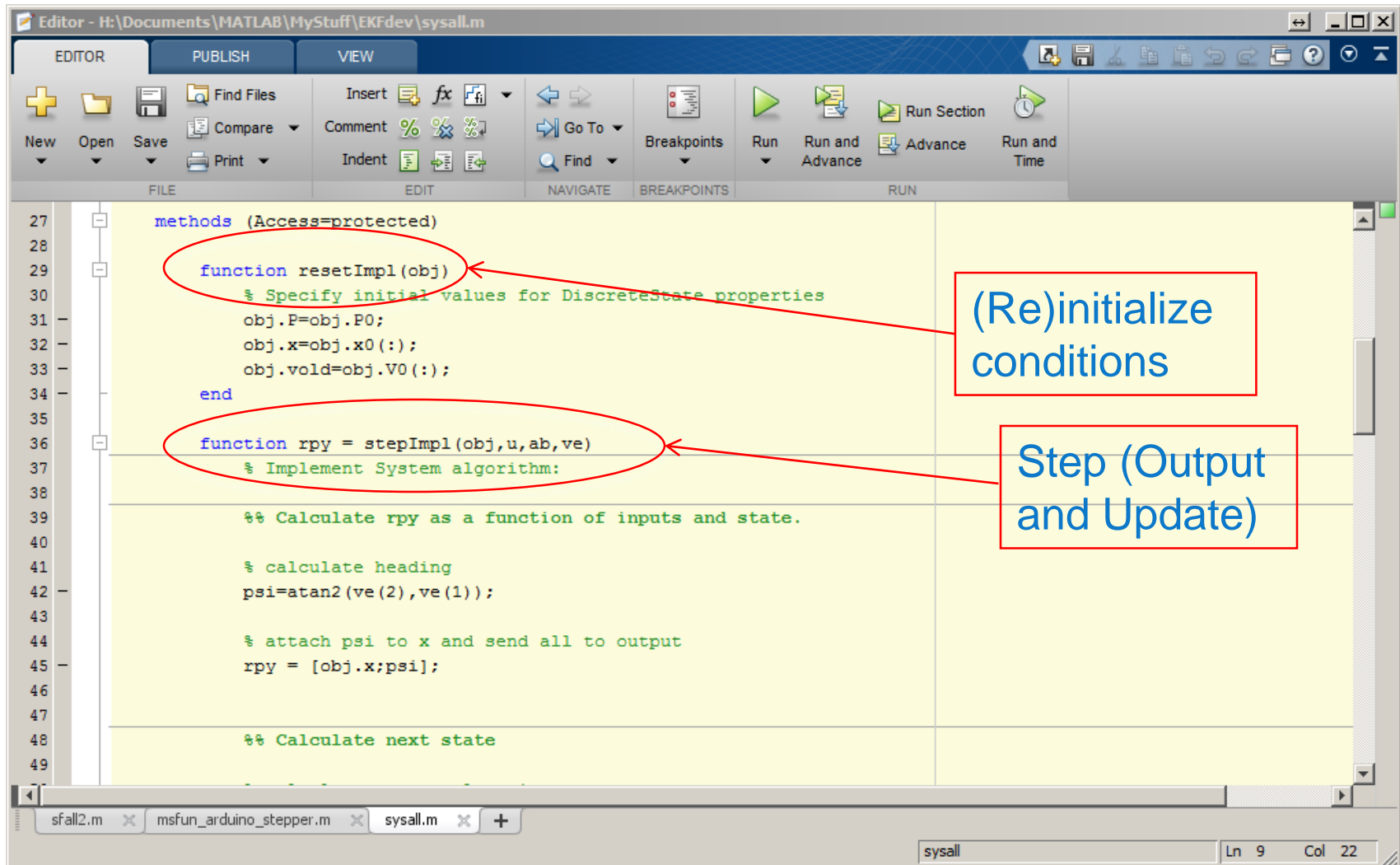
```

Object definition

Parameters (to be passed from Simulink)

Discrete States

MATLAB System object block



```

27 methods (Access=protected)
28
29 function resetImpl(obj)
30     % Specify initial values for DiscreteState properties
31     obj.P=obj.P0;
32     obj.x=obj.x0(:);
33     obj.vold=obj.V0(:);
34 end
35
36 function rpy = stepImpl(obj,u,ab,ve)
37     % Implement System algorithm:
38
39     %% Calculate rpy as a function of inputs and state.
40
41     % calculate heading
42     psi=atan2(ve(2),ve(1));
43
44     % attach psi to x and send all to output
45     rpy = [obj.x;psi];
46
47
48     %% Calculate next state
49

```

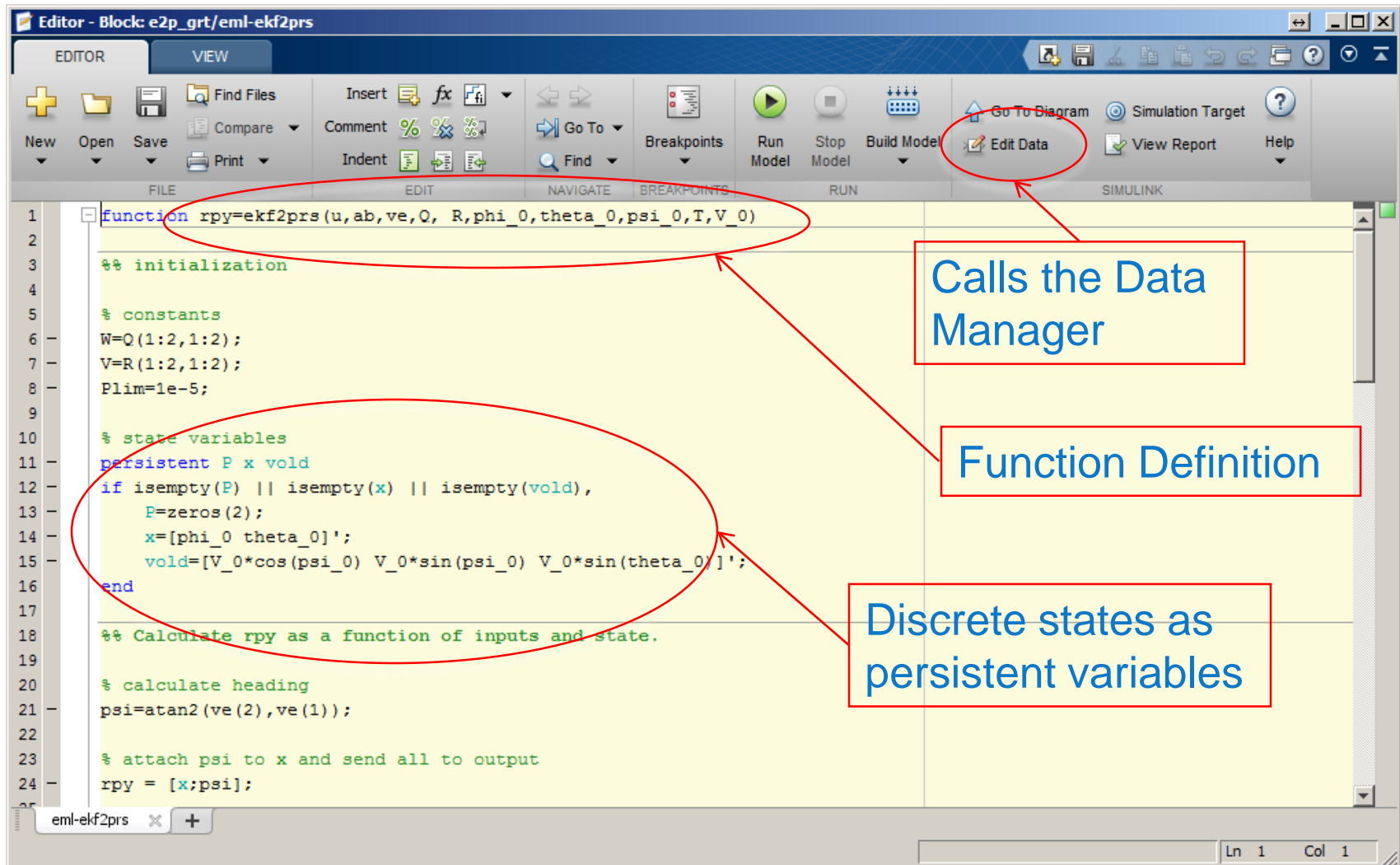
(Re)initialize conditions

Step (Output and Update)

MATLAB System object block: pros and cons

- The API is **simpler and more elegant** than S-functions.
- Allows **code generation** (in simulations and can be executed both in interpreted or compiled mode).
- The mask is generated automatically. Allows for discrete state properties, and it's easier to work with external libraries.
- Allows for **MATLAB-only** (no Simulink) simulations!
- Relies more heavily on OO concepts. Constrained structure may feel too rigid for “one offs”, but is good for systematic development, deployment and maintenance.

MATLAB function block



The screenshot shows the MATLAB Editor interface with a function block named `eml-ekf2prs`. The function definition is as follows:

```

1 function rpy=ekf2prs(u,ab,ve,Q, R,phi_0,theta_0,psi_0,T,V_0)
2
3 %% initialization
4
5 % constants
6 W=Q(1:2,1:2);
7 V=R(1:2,1:2);
8 Plim=1e-5;
9
10 % state variables
11 persistent P x vold
12 if isempty(P) || isempty(x) || isempty(vold),
13     P=zeros(2);
14     x=[phi_0 theta_0]';
15     vold=[V_0*cos(psi_0) V_0*sin(psi_0) V_0*sin(theta_0)]';
16 end
17
18 %% Calculate rpy as a function of inputs and state.
19
20 % calculate heading
21 psi=atan2(ve(2),ve(1));
22
23 % attach psi to x and send all to output
24 rpy = [x;psi];
25

```

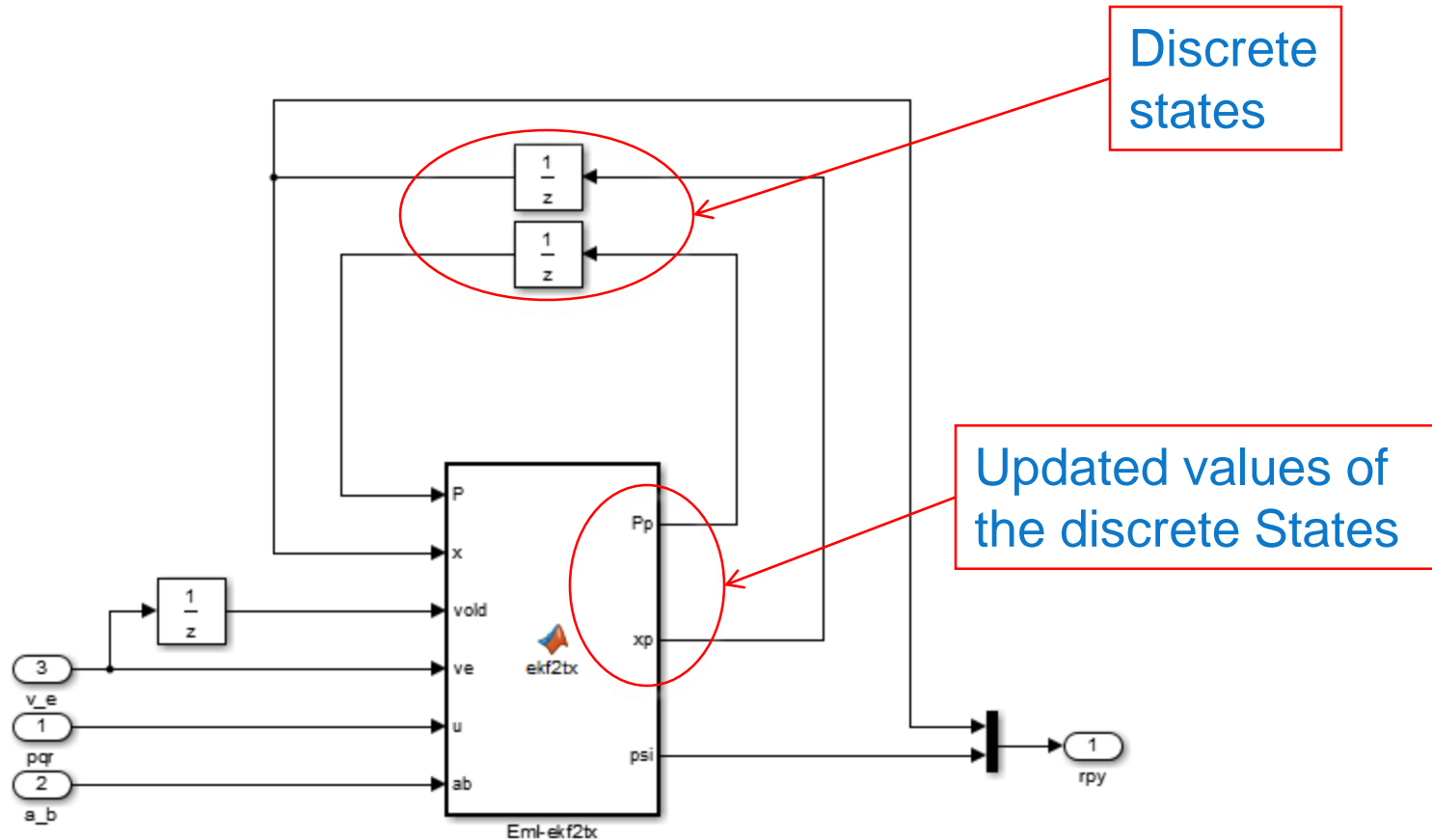
Annotations in the image highlight specific features:

- Go To Diagram**: A button in the top toolbar, circled in red, with an arrow pointing to the text "Calls the Data Manager".
- Function Definition**: A red oval around the function signature `function rpy=ekf2prs(u,ab,ve,Q, R,phi_0,theta_0,psi_0,T,V_0)` with an arrow pointing to the text "Function Definition".
- Discrete states as persistent variables**: A red oval around the persistent variable declaration `persistent P x vold` and the initialization logic, with an arrow pointing to the text "Discrete states as persistent variables".

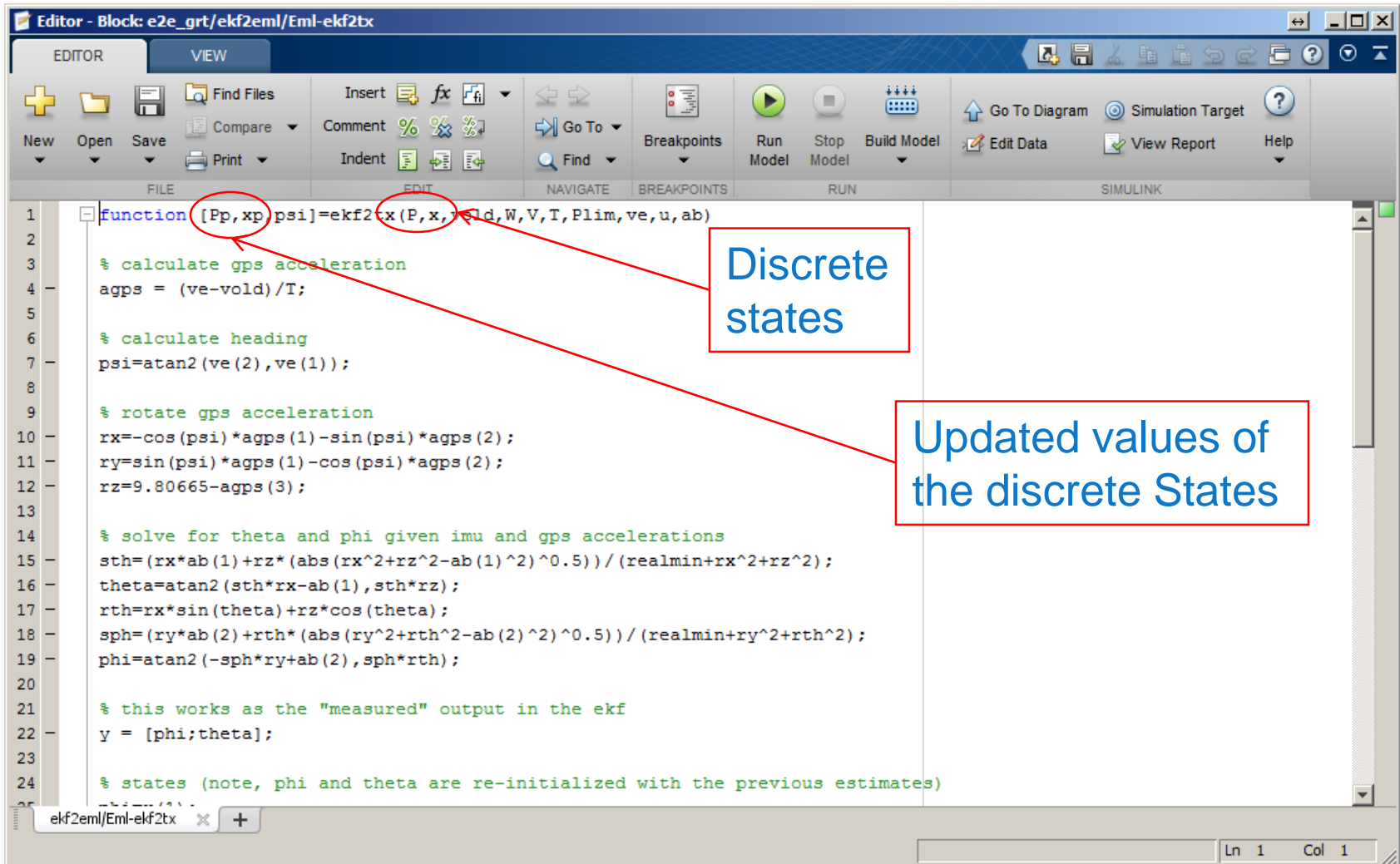
MATLAB function block: pros and cons

- Perhaps the **simplest** method once you know how to use the Data Manager.
- Allows **code generation**
- The default mask is not too descriptive, so a more descriptive mask must be manually added if needed.
- The lack of structure allows for a lot of flexibility and potentially simplifies things.
- Good for “one-off” implementations.

MATLAB function with external states



MATLAB function with external states



The image shows a MATLAB Editor window with a function file named `ekf2eml/Eml-ekf2tx`. The function signature is `function [Pp, xp, psi]=ekf2tx(P, x, v, d, W, V, T, Plim, ve, u, ab)`. Two red circles highlight the inputs `x` and `psi`. Red arrows point from these circles to two text boxes: "Discrete states" pointing to `x` and "Updated values of the discrete States" pointing to `psi`.

```

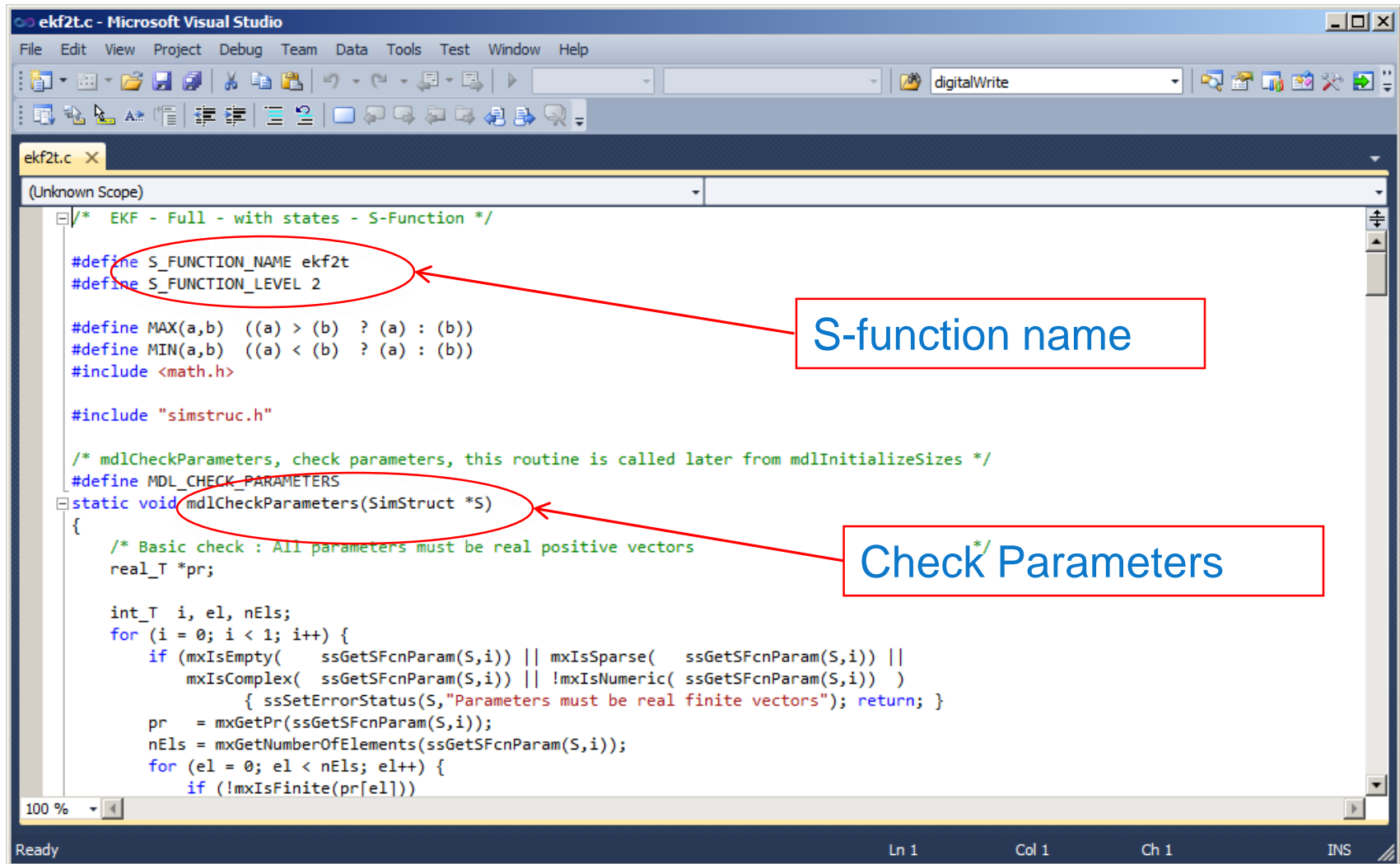
1 function [Pp, xp, psi]=ekf2tx(P, x, v, d, W, V, T, Plim, ve, u, ab)
2
3 % calculate gps acceleration
4 agps = (ve-vold)/T;
5
6 % calculate heading
7 psi=atan2(ve(2),ve(1));
8
9 % rotate gps acceleration
10 rx=-cos(psi)*agps(1)-sin(psi)*agps(2);
11 ry=sin(psi)*agps(1)-cos(psi)*agps(2);
12 rz=9.80665-agps(3);
13
14 % solve for theta and phi given imu and gps accelerations
15 sth=(rx*ab(1)+rz*(abs(rx^2+rz^2-ab(1)^2)^0.5)/(realmin+rx^2+rz^2);
16 theta=atan2(sth*rx-ab(1),sth*rz);
17 rth=rx*sin(theta)+rz*cos(theta);
18 sph=(ry*ab(2)+rth*(abs(ry^2+rth^2-ab(2)^2)^0.5)/(realmin+ry^2+rth^2);
19 phi=atan2(-sph*ry+ab(2),sph*rth);
20
21 % this works as the "measured" output in the ekf
22 y = [phi;theta];
23
24 % states (note, phi and theta are re-initialized with the previous estimates)
25

```

MATLAB function with external states: pros and cons

- Is a more structured way of implementing the algorithm in which the states are externally held by unit delays and therefore clearly visible. This simplifies the MATLAB Function.
- Is only here for comparison purposes, probably not worth the extra work with respect to the previous method.
- However it might be useful to implement **continuous time** algorithms. This can be done by using integrators instead of unit delays and calculating (in the MATLAB function block) the state derivative instead of the state update.

C S-function (level 2)



```

ekf2t.c - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Test Window Help

ekf2t.c
(Unknown Scope)

/* EKF - Full - with states - S-Function */

#define S_FUNCTION_NAME ekf2t
#define S_FUNCTION_LEVEL 2

#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
#include <math.h>

#include "simstruc.h"

/* mdlCheckParameters, check parameters, this routine is called later from mdlInitializeSizes */
#define MDL_CHECK_PARAMETERS
static void mdlCheckParameters(SimStruct *S)
{
    /* Basic check : All parameters must be real positive vectors
    real_T *pr;

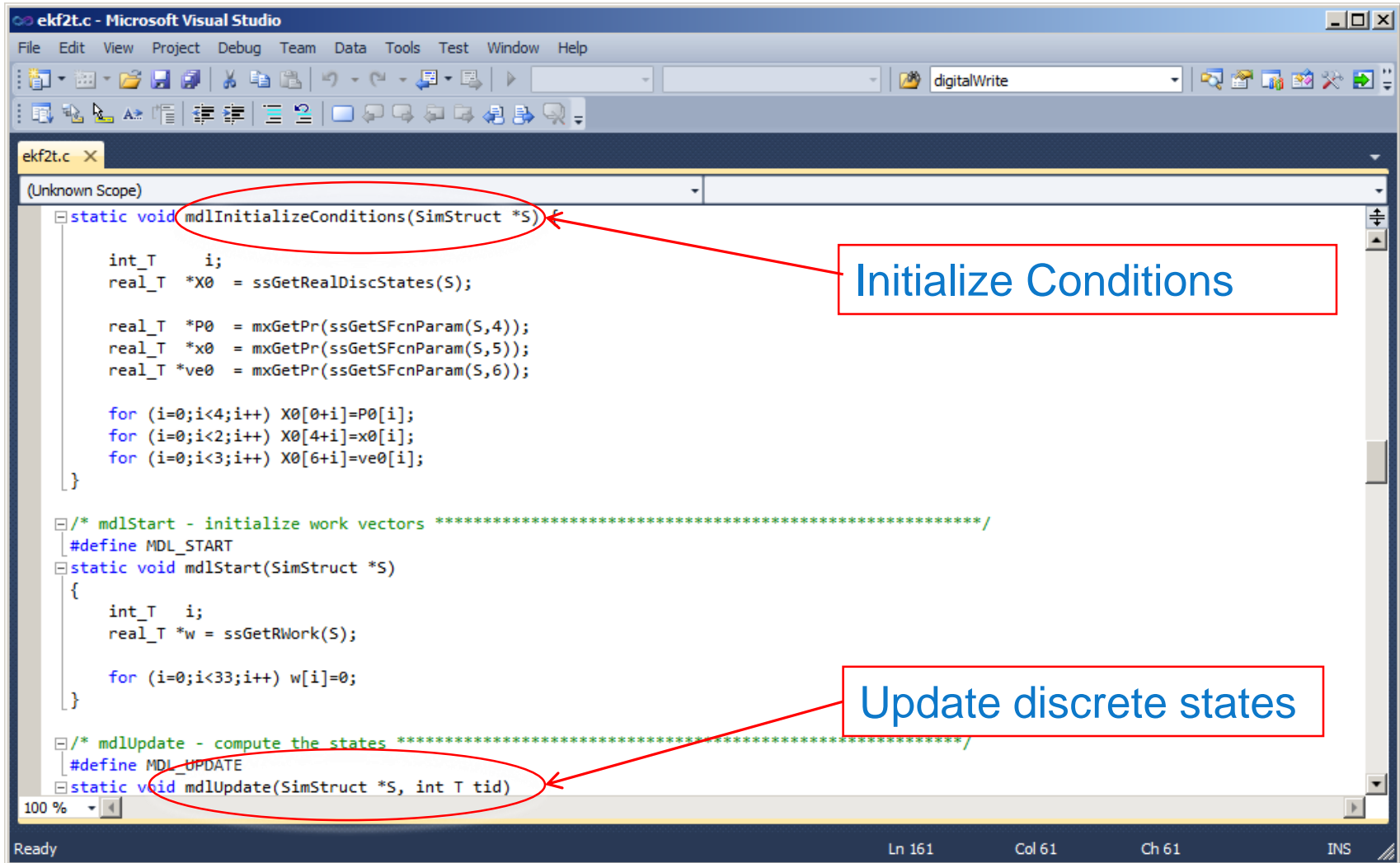
    int_T i, el, nEls;
    for (i = 0; i < 1; i++) {
        if (mxIsEmpty( ssGetSFcnParam(S,i)) || mxIsSparse( ssGetSFcnParam(S,i)) ||
            mxIsComplex( ssGetSFcnParam(S,i)) || !mxIsNumeric( ssGetSFcnParam(S,i)) )
            { ssSetErrorStatus(S,"Parameters must be real finite vectors"); return; }
        pr = mxGetPr(ssGetSFcnParam(S,i));
        nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
        for (el = 0; el < nEls; el++) {
            if (!mxIsFinite(pr[el]))

```

S-function name

Check* Parameters

C S-function (level 2)



```

ekf2t.c - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Test Window Help

ekf2t.c x
(Unknown Scope)
static void mdlInitializeConditions(SimStruct *S)
{
    int_T i;
    real_T *X0 = ssGetRealDiscStates(S);

    real_T *P0 = mxGetPr(ssGetSFcnParam(S,4));
    real_T *x0 = mxGetPr(ssGetSFcnParam(S,5));
    real_T *ve0 = mxGetPr(ssGetSFcnParam(S,6));

    for (i=0;i<4;i++) X0[0+i]=P0[i];
    for (i=0;i<2;i++) X0[4+i]=x0[i];
    for (i=0;i<3;i++) X0[6+i]=ve0[i];
}

/* mdlStart - initialize work vectors *****/
#define MDL_START
static void mdlStart(SimStruct *S)
{
    int_T i;
    real_T *w = ssGetRWork(S);

    for (i=0;i<33;i++) w[i]=0;
}

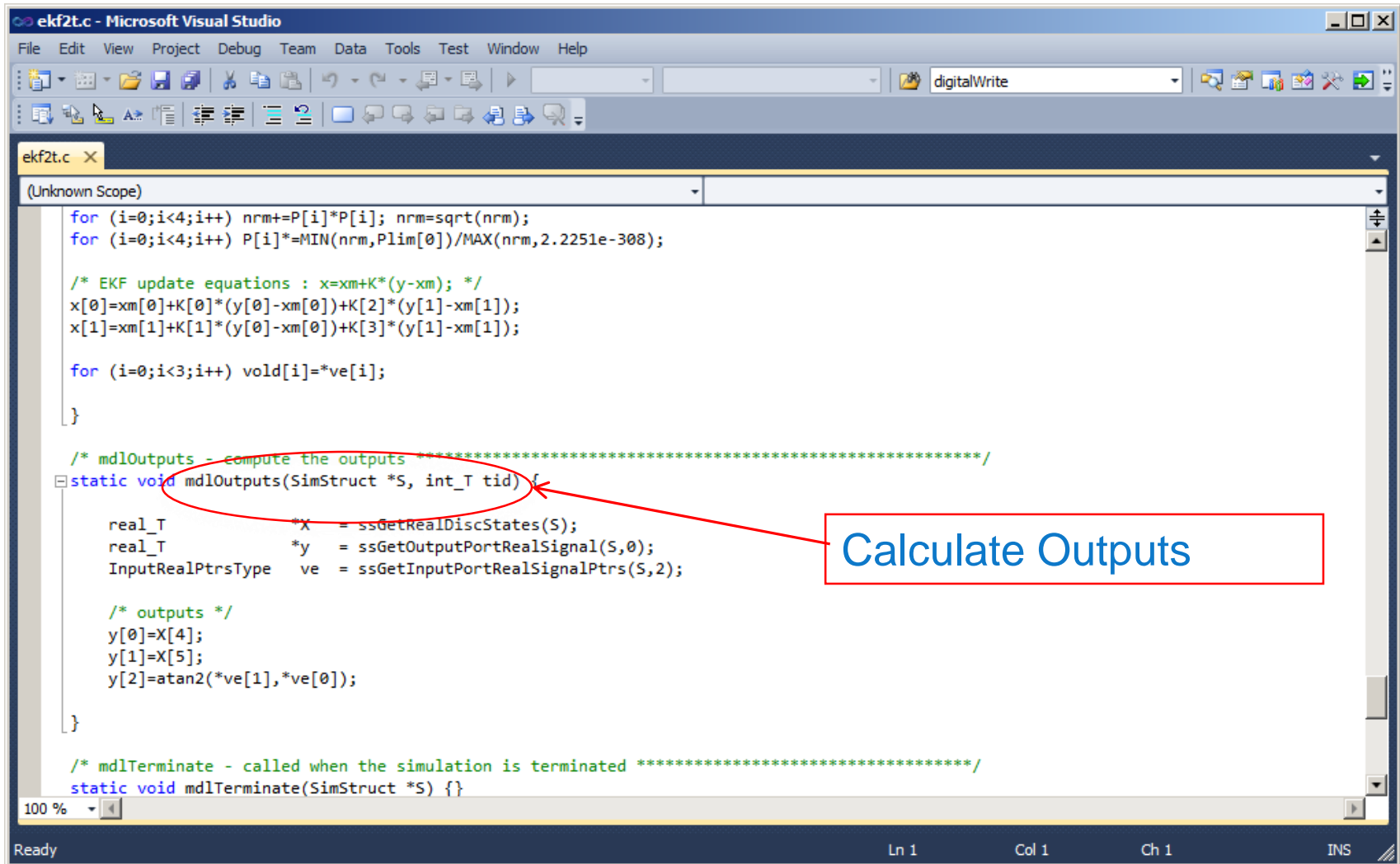
/* mdlUpdate - compute the states *****/
#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int T tid)

```

Initialize Conditions

Update discrete states

C S-function (level 2)



```

ekf2t.c - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Test Window Help

for (i=0;i<4;i++) nrm+=P[i]*P[i]; nrm=sqrt(nrm);
for (i=0;i<4;i++) P[i]=MIN(nrm,Plim[0])/MAX(nrm,2.2251e-308);

/* EKF update equations : x=xm+K*(y-xm); */
x[0]=xm[0]+K[0]*(y[0]-xm[0])+K[2]*(y[1]-xm[1]);
x[1]=xm[1]+K[1]*(y[0]-xm[0])+K[3]*(y[1]-xm[1]);

for (i=0;i<3;i++) void[i]=*ve[i];
}

/* mdlOutputs - compute the outputs *****/
static void mdlOutputs(SimStruct *S, int_T tid) {
    real_T *X = ssGetRealDiscStates(S);
    real_T *y = ssGetOutputPortRealSignal(S,0);
    InputRealPtrsType ve = ssGetInputPortRealSignalPtrs(S,2);

    /* outputs */
    y[0]=X[4];
    y[1]=X[5];
    y[2]=atan2(*ve[1],*ve[0]);
}

/* mdlTerminate - called when the simulation is terminated *****/
static void mdlTerminate(SimStruct *S) {}
    
```

Calculate Outputs

C S-function : pros and cons

- Supports SimStruct and the entire S-function API (in some sense it is even more powerful than MATLAB S-functions).
- **Is compiled.**
- It must be **handwritten in C** (not feasible for large algorithms requiring linear algebra and/or MATLAB toolboxes functions).
- **Allows code generation only for targets supporting non-inlined S-functions** (unless you write a TLC file).

S-Function Builder

S-Function Builder: e2b_grt/S-Function Builder

Parameters

S-function name:

S-function parameters

Name	Data type	Value
W	double	$1e-5 * [0.1134 \ 0 \ 0 \ 0.312]$
V	double	$[0.3368 \ 0 \ 0 \ 0.0920]$
Plim	double	$1e-5$

Port/Parameter

- Input Ports
 - pqr
 - a_b
 - v_e
- Output Ports
 - rpy
- Parameters
 - W
 - V
 - Plim

Initialization | Data Properties | Libraries | Outputs | Continuous Derivatives | Discrete Update | Build Info

Description

The S-Function Builder block creates a wrapper C-MEX S-function from your supplied C code with multiple input ports, output ports, and a variable number of scalar, vector, or matrix parameters. The input and output ports can propagate Simulink built-in data types, fixed-point datatypes, complex, frame, 1-D, and 2-D signals. This block also supports discrete and continuous states of type real. You can optionally have the block generate a TLC file to be used with Simulink Coder for code generation.

S-function settings

Number of discrete states: Sample mode:

Discrete states IC: Sample time value:

Number of continuous states:

Continuous states IC:

Parameters

Initial Conditions
and Sample Time

Must be constants
cannot be variables,
(rebuild necessary if
they are changed).

S-Function Builder

S-Function Builder: e2b_grt/S-Function Builder

Parameters

S-function name: Build

S-function parameters

Name	Data type	Value
W	double	1e-5*[0.1134 0 0 0.312]
V	double	[0.3368 0 0 0.0920]
Plim	double	1e-5

Port/Parameter

- Input Ports
 - pqr
 - a_b
 - v_e
- Output Ports
 - rpy
- Parameters
 - W
 - V
 - Plim

Initialization | Data Properties | Libraries | **Outputs** | Continuous Derivatives | Discrete Update | Build Info

Code description

Enter your C-code or call your algorithm. If available, discrete and continuous states should be referenced as, xD[0]...xD[n], xC[0]...xC[n] respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```

/* outputs */
rpy[0]=xD[4];
rpy[1]=xD[5];
rpy[2]=atan2(v_e[1],v_e[0]);
  
```

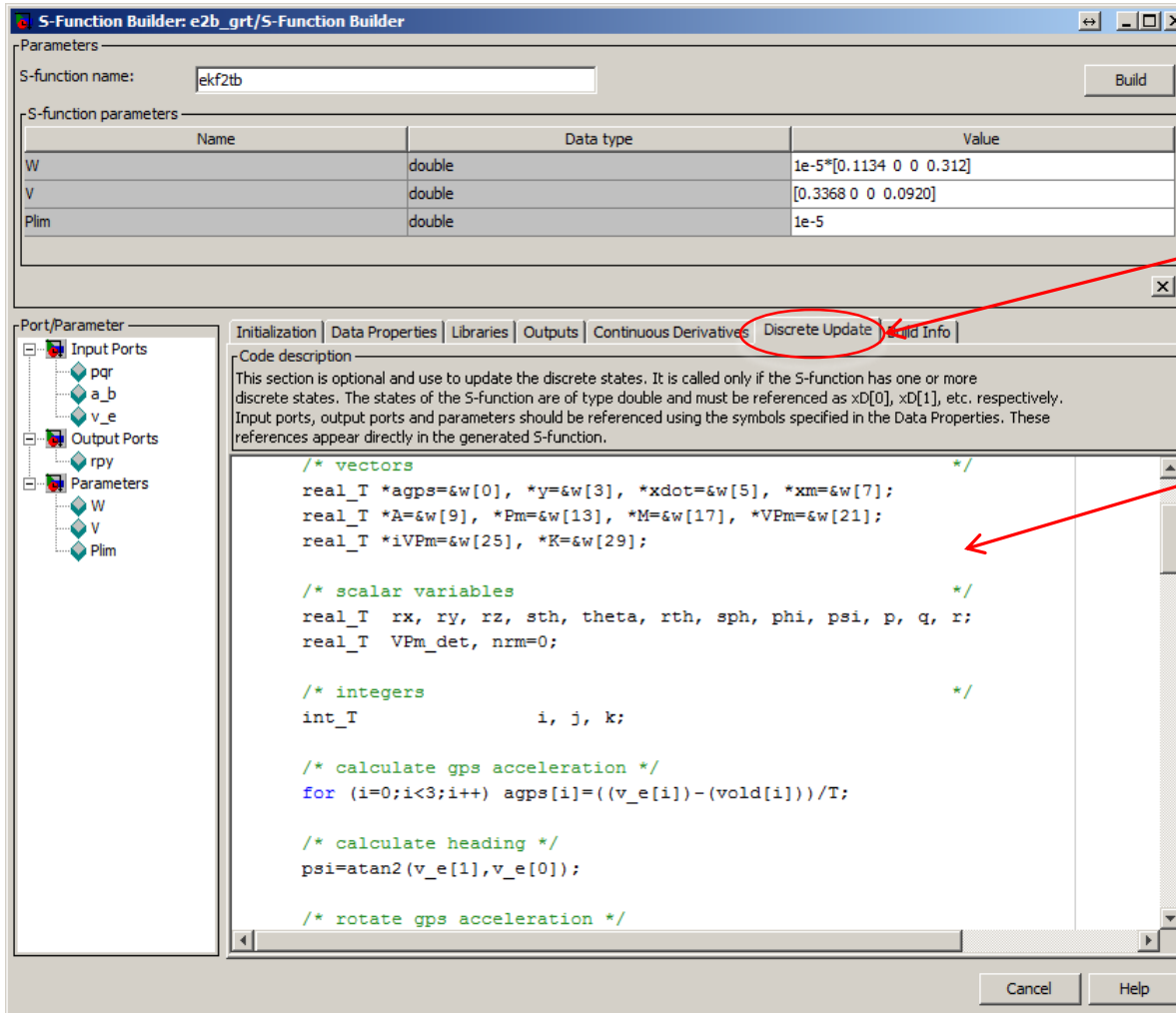
☒ Inputs are needed in the output function(direct feedthrough)

Cancel Help

Outputs pane

Outputs calculation
(xD is the vector of
discrete states and
work variables)

S-Function Builder



S-Function Builder: e2b_grt/S-Function Builder

Parameters

S-function name: Build

S-function parameters

Name	Data type	Value
W	double	1e-5*[0.1134 0 0 0.312]
V	double	[0.3368 0 0 0.0920]
Plim	double	1e-5

Port/Parameter

- Input Ports
 - pqr
 - a_b
 - v_e
- Output Ports
 - rpy
- Parameters
 - W
 - V
 - Plim

Initialization | Data Properties | Libraries | Outputs | Continuous Derivatives | **Discrete Update** | Build Info

Code description

This section is optional and use to update the discrete states. It is called only if the S-function has one or more discrete states. The states of the S-function are of type double and must be referenced as xD[0], xD[1], etc. respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```

/* vectors */
real_T *agps=&w[0], *y=&w[3], *xdot=&w[5], *xm=&w[7];
real_T *A=&w[9], *Pm=&w[13], *M=&w[17], *VPm=&w[21];
real_T *iVPm=&w[25], *K=&w[29];

/* scalar variables */
real_T rx, ry, rz, sth, theta, rth, sph, phi, psi, p, q, r;
real_T VPm_det, nrm=0;

/* integers */
int_T i, j, k;

/* calculate gps acceleration */
for (i=0;i<3;i++) agps[i]=((v_e[i])-(vold[i]))/T;

/* calculate heading */
psi=atan2(v_e[1],v_e[0]);

/* rotate gps acceleration */

```

Cancel Help

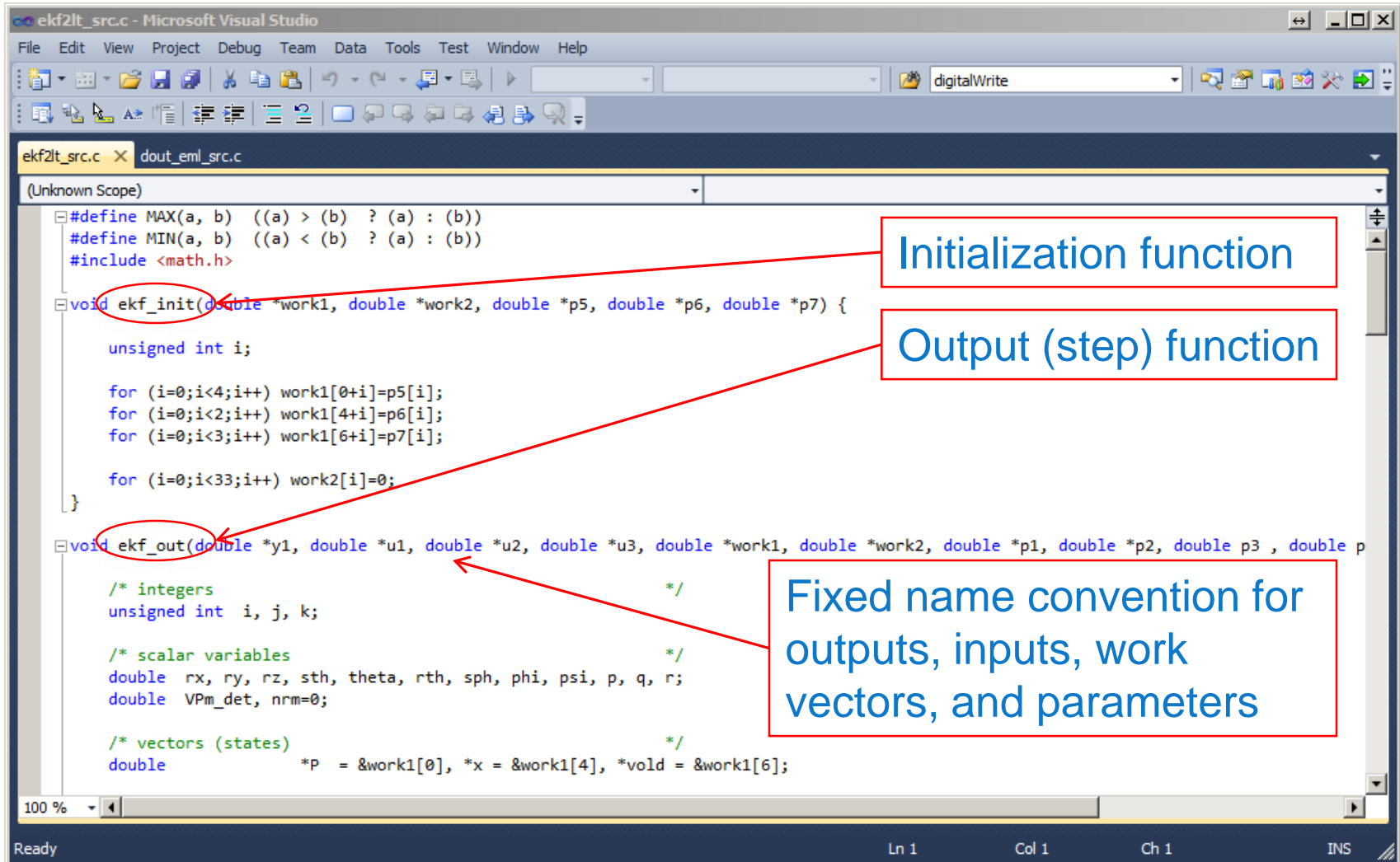
Update pane

Calculation of the update for the discrete states.

S-Function Builder: pros and cons

- Less flexible than handwritten S-function. Initial states and sample time **cannot be passed as parameters**. Also, masks are handled differently than other blocks.
- It is compiled. However the generated S-function code uses a wrapper function, which causes a small additional overhead in simulation mode.
- The builder automatically generates a TLC file, therefore it **allows code generation for any target**.
- It still requires some C and S-function knowledge. Initialization must be performed through update function.

Legacy Code Tool: the C code



```

ekf2lt_src.c - Microsoft Visual Studio
File Edit View Project Debug Team Data Tools Test Window Help

ekf2lt_src.c x dout_eml_src.c
(Unknown Scope)

#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define MIN(a, b) ((a) < (b) ? (a) : (b))
#include <math.h>

void ekf_init(double *work1, double *work2, double *p5, double *p6, double *p7) {
    unsigned int i;

    for (i=0;i<4;i++) work1[0+i]=p5[i];
    for (i=0;i<2;i++) work1[4+i]=p6[i];
    for (i=0;i<3;i++) work1[6+i]=p7[i];

    for (i=0;i<33;i++) work2[i]=0;
}

void ekf_out(double *y1, double *u1, double *u2, double *u3, double *work1, double *work2, double *p1, double *p2, double p3, double p
    /* integers */
    unsigned int i, j, k;

    /* scalar variables */
    double rx, ry, rz, sth, theta, rth, sph, phi, psi, p, q, r;
    double VPM_det, nrm=0;

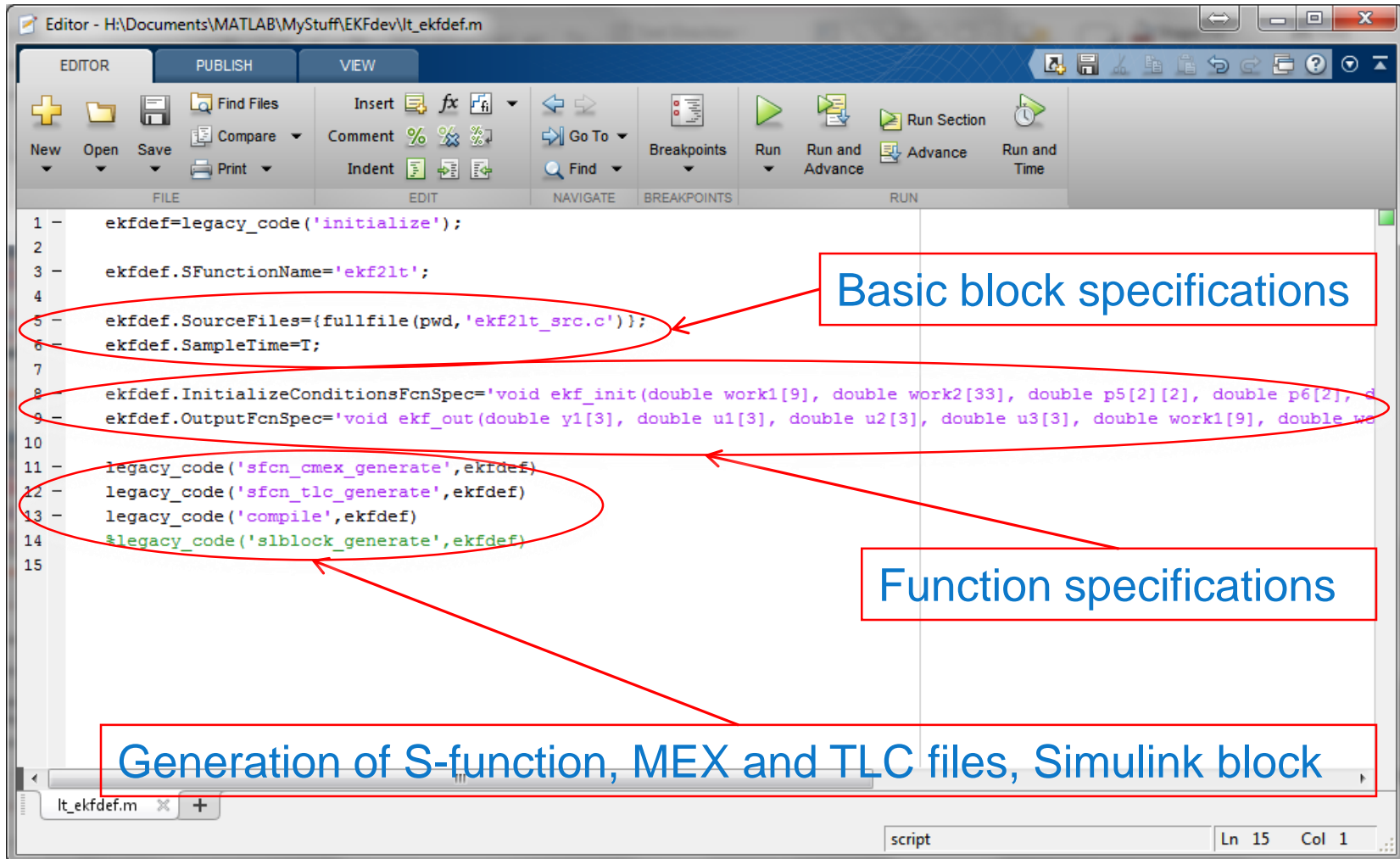
    /* vectors (states) */
    double *P = &work1[0], *x = &work1[4], *vold = &work1[6];
    
```

Initialization function

Output (step) function

Fixed name convention for outputs, inputs, work vectors, and parameters

Legacy Code Tool: assembling the block



```

1 - ekfdef=legacy_code('initialize');
2 -
3 - ekfdef.SFunctionName='ekf2lt';
4 -
5 - ekfdef.SourceFiles={fullfile(pwd,'ekf2lt_src.c')};
6 - ekfdef.SampleTime=T;
7 -
8 - ekfdef.InitializeConditionsFcnSpec='void ekf_init(double work1[9], double work2[33], double p5[2][2], double p6[2], d
9 - ekfdef.OutputFcnSpec='void ekf_out(double yl[3], double ul[3], double u2[3], double u3[3], double work1[9], double w
10 -
11 - legacy_code('sfcn_cmex_generate',ekfdef)
12 - legacy_code('sfcn_tlc_generate',ekfdef)
13 - legacy_code('compile',ekfdef)
14 - %legacy_code('slblock_generate',ekfdef)
15 -

```

Basic block specifications

Function specifications

Generation of S-function, MEX and TLC files, Simulink block

Legacy Code Tool: pros and cons

- Completely programmatic interface (no GUI) oriented towards the integration of existing C code.
- It is compiled. It does not use any wrapper. Supports less features than the S-Function Builder.
- S-function and TLC files are automatically generated. **Code generation is allowed for any target** and optimized for faster execution on embedded systems.
- It still requires some C knowledge (but no S-function knowledge).



Pure Simulink

- Only Simulink knowledge required.
- It is compiled.
- S-function, MEX and TLC files are not required. Only one model file required thus easy to ship. **Code generation is allowed for any target.**
- Harder for large algorithms requiring either linear algebra, a lot of logic, and/or MATLAB toolbox functions. Harder to deal with the initialization function.

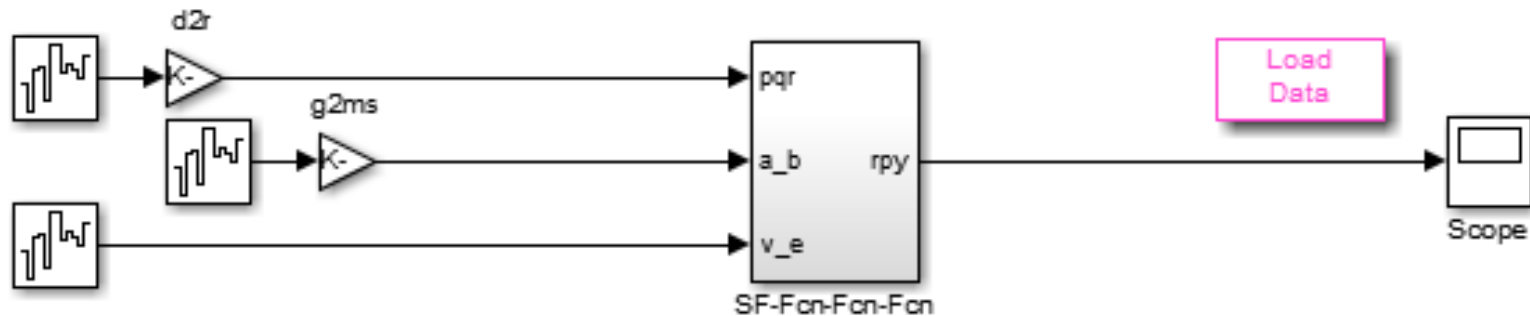
Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

Informal performance comparison

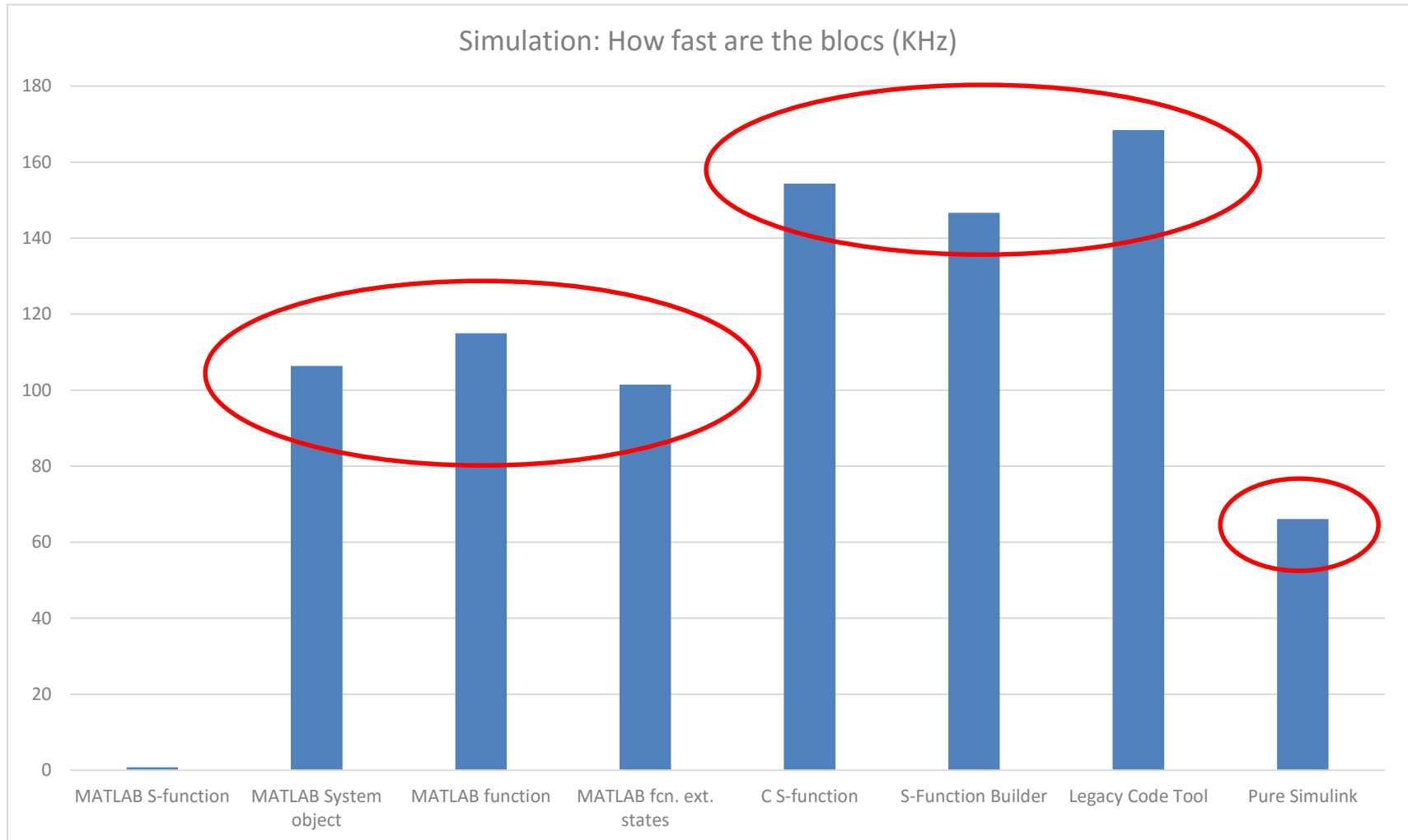
- Simulink blocks were created using the methods previously described (one for each method).
- Simulations were then run to verify that blocks reproduced the same outputs from the same inputs, and starting from the same initial conditions.
- Simple simulations (containing just a source and the EKF blocks, see next page) were then run programmatically (each one multiple times) in MATLAB 2016b on an Intel Xeon L5630, 2.13GHz, 2-Cores, 8GB RAM, Win7, 64bit laptop.

Example model for performance comparison

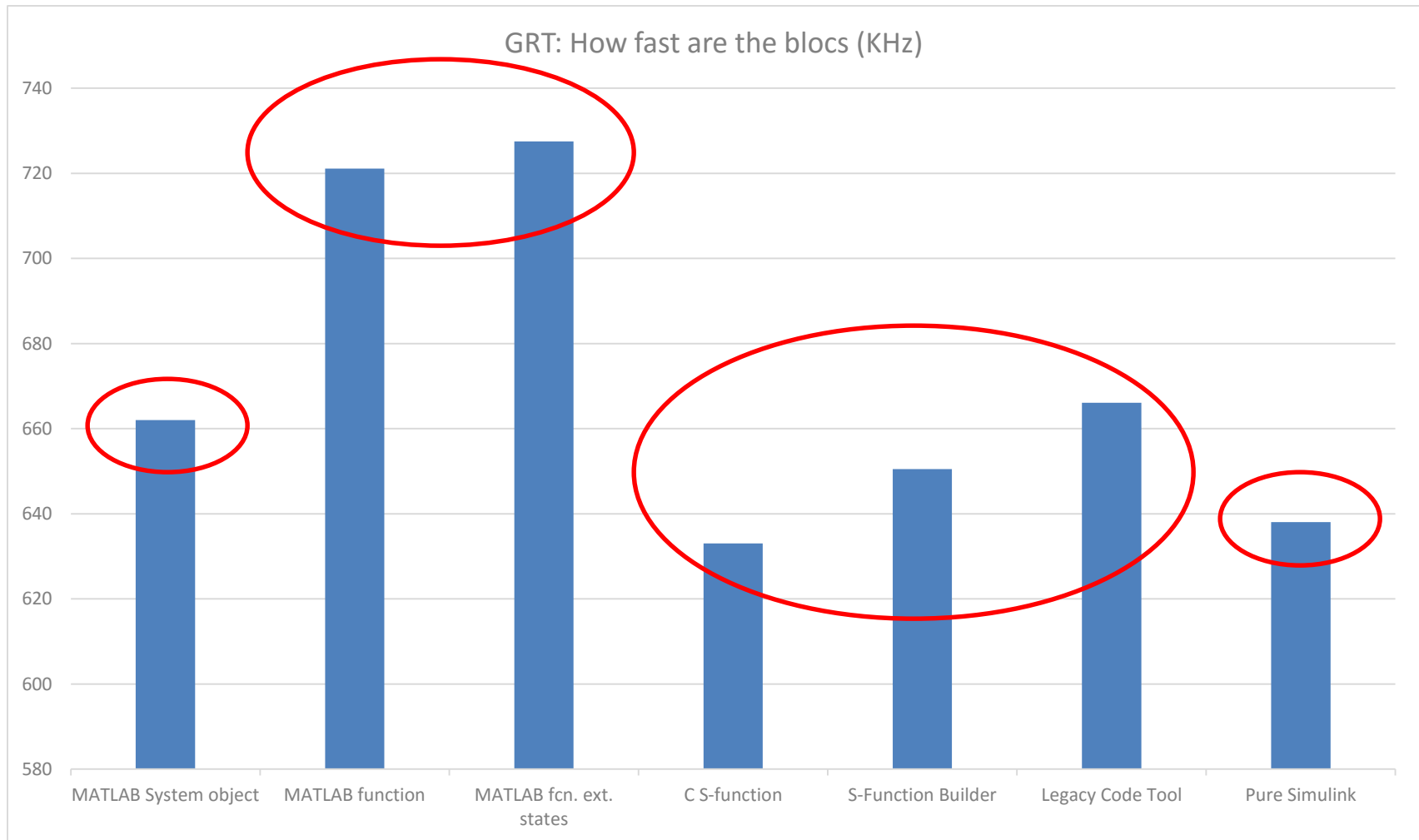


- Simulation time was set to $1e5$ seconds, and the sampling time was 0.05 seconds.
- Elapsed time was measured using `tic` and `toc`, and averaged over 4 different executions (so, not rigorous).
- Maximum achievable frequency was calculated dividing the number of steps ($1e5/0.05$) by the elapsed time.

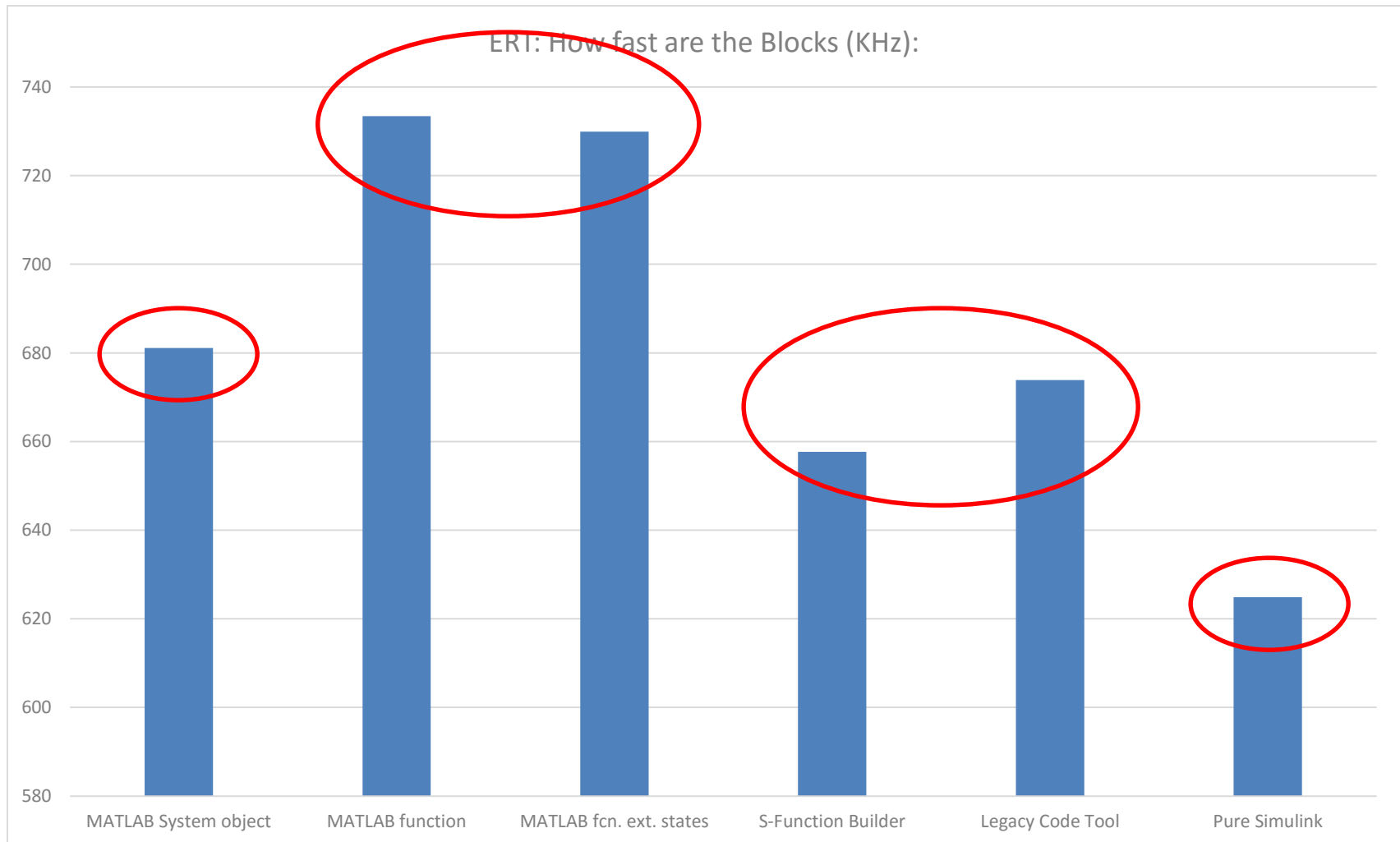
Simulation only



GRT executables

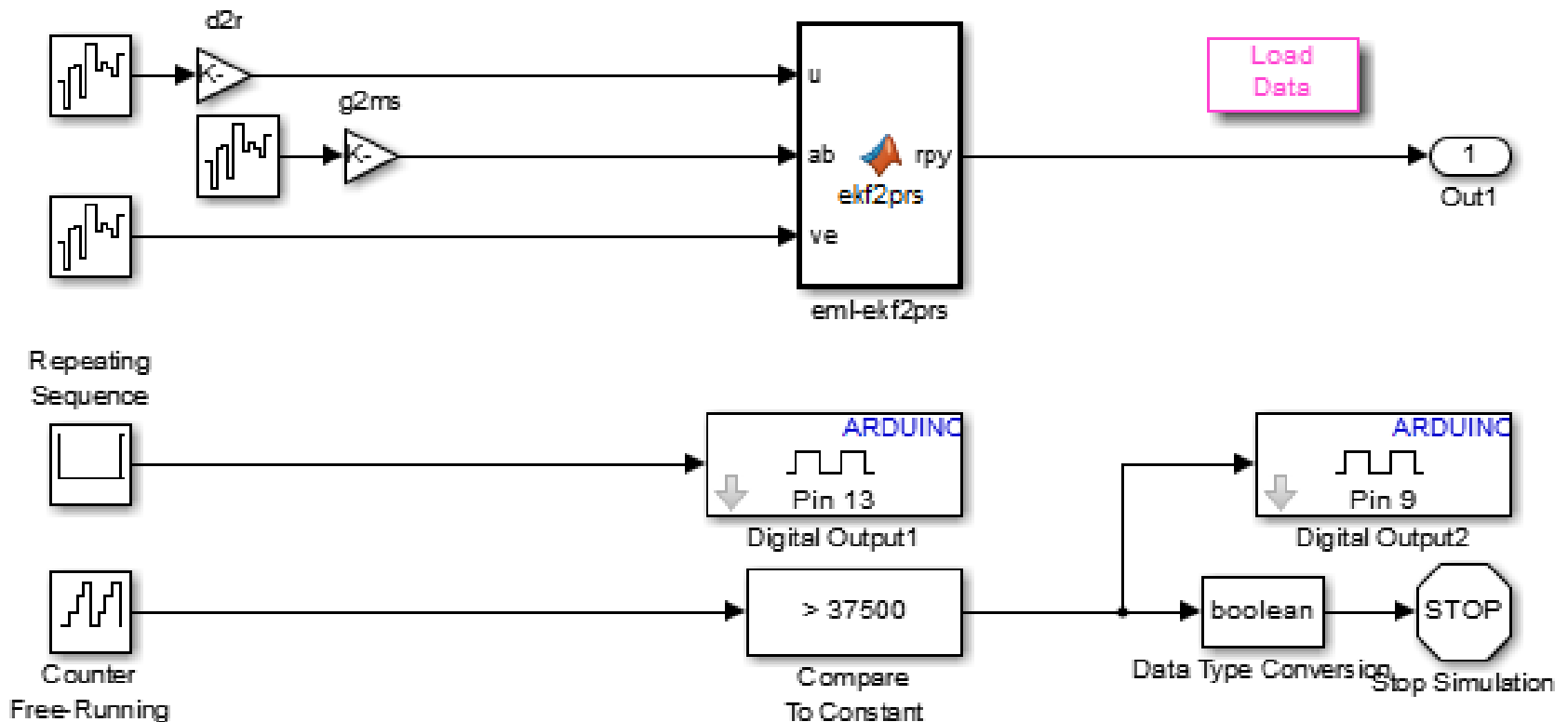


ERT executables



Arduino Uno

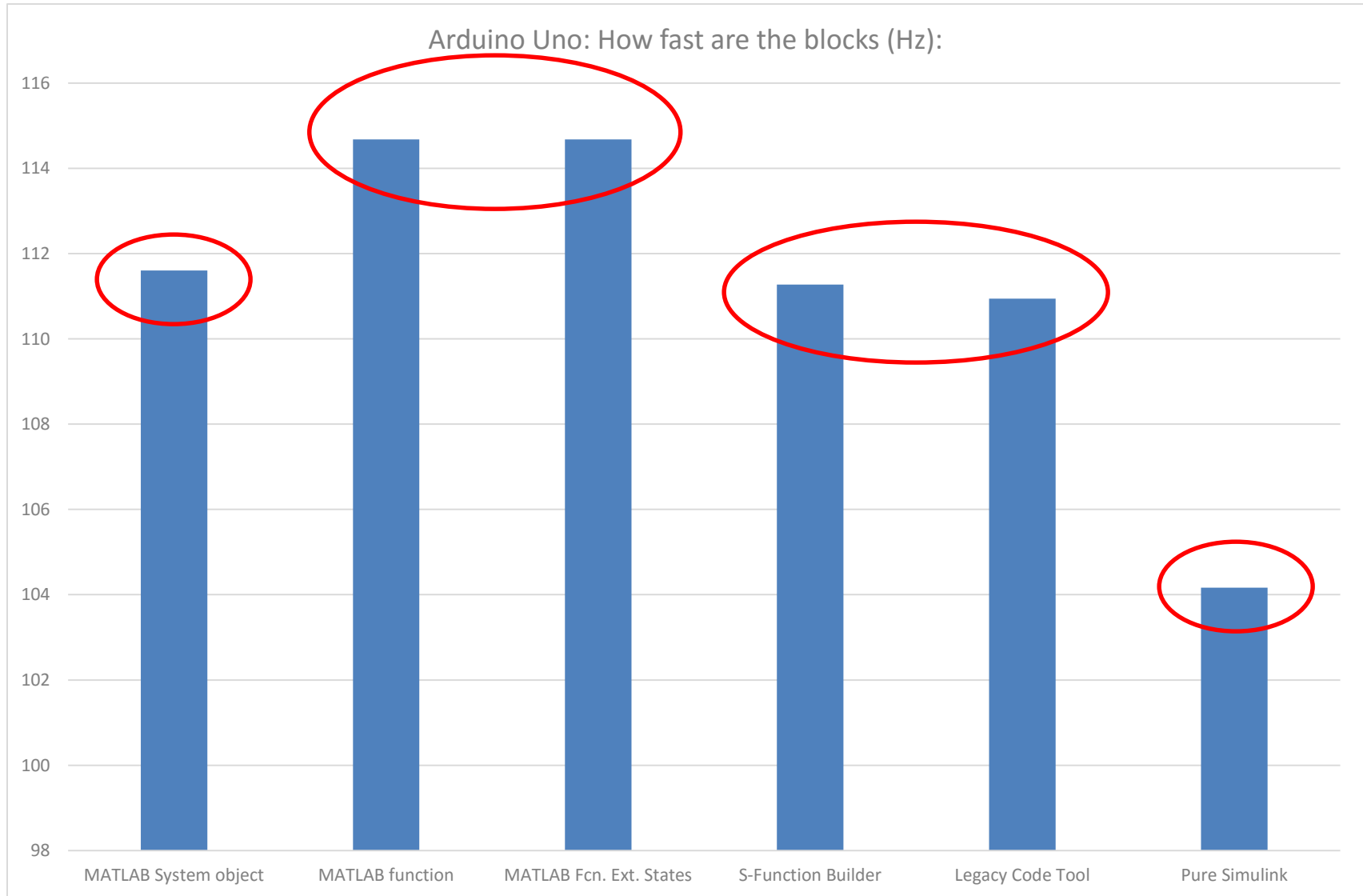
The previous Simulink models were augmented with digital output blocks to light up a LED after 5 minutes:



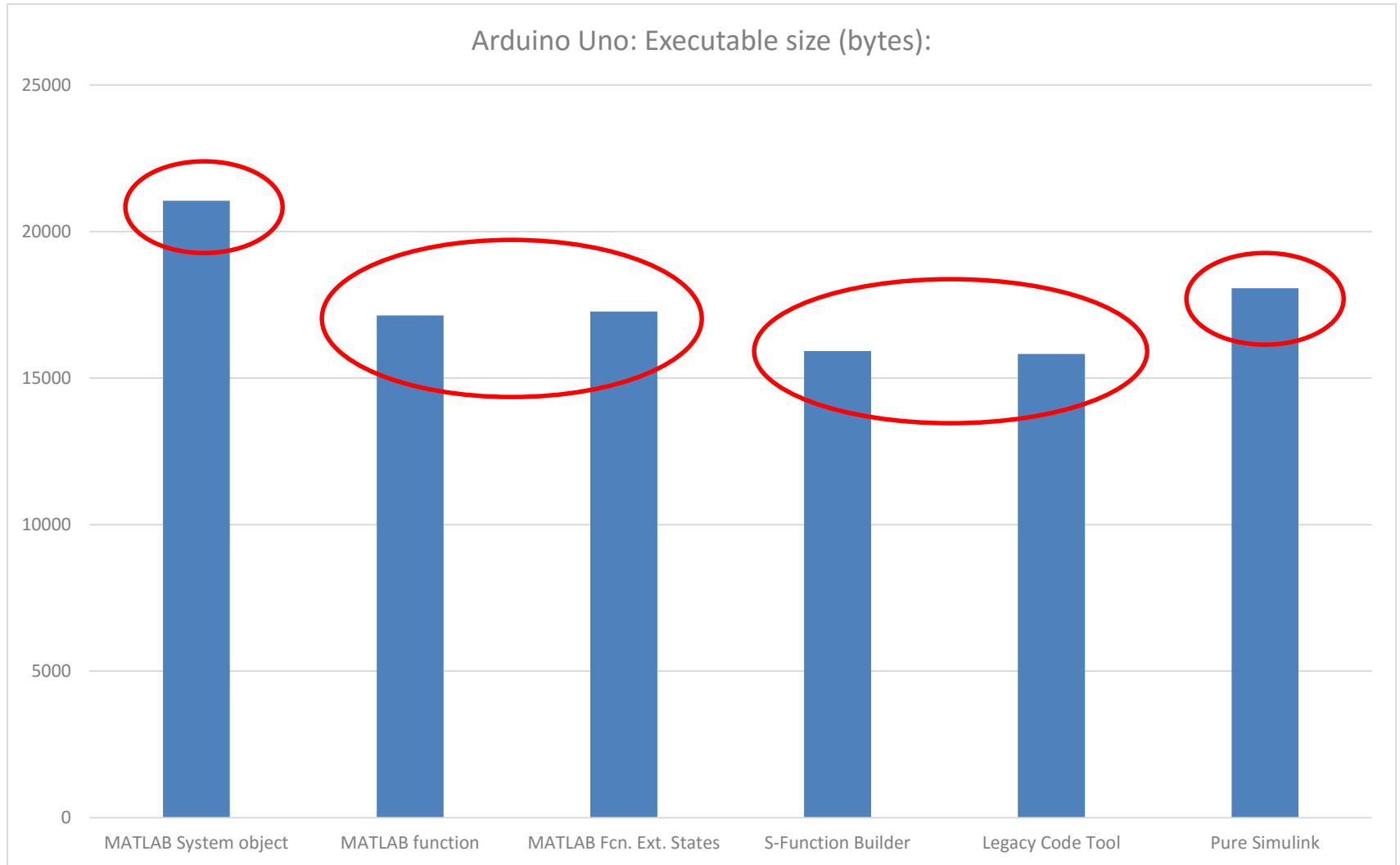
Arduino Uno

- Up until a sampling frequency of 100Hz the execution was fine, and the LED on pin 9 actually lit up after exactly 300s (as measured with a stopwatch).
- Whenever the frequency was pushed to 125Hz (base sample time $T=0.008$ sec) the different executables started to accumulate different delays (so termination happened 30-60s later than 5 minutes).
- Maximum achievable frequency was calculated dividing the number of steps ($300/0.008$) by the total elapsed time (e.g. 337 sec for the S-Function Builder block)

Arduino Uno



Arduino Uno



Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- **Conclusions**

Conclusions

- MATLAB System object, MATLAB function, S-Function Builder, Legacy Code Tool and pure Simulink work for any kind of target.
- The performance comparison was somewhat informal (and compiler-dependent), however:
 - **Methods based on C tend to be faster only in simulation**
 - **Methods based on MATLAB tend to be faster for on-target execution.**
 - The MATLAB Function block is marginally faster than the System Object block, and is OK for one-offs, but the latter has many features that make it easier to develop, deploy and maintain a block.